Developing General Video Game Playing Artificial Intelligence Based on a Multilayer Architecture

VALLADE BENOIT

June 2016

Doctoral Thesis at Osaka Prefecture University

Acknowledgement

I would like to express my deepest appreciation and gratitude to Professor Tomoharu Nakashima, my laboratory and dissertation supervisor, whose guidance, advice, and constant concern gave me the opportunity to complete this thesis.

Furthermore I am deeply grateful to the members of my research team, Alexandre David, Masahiro Hori, Takuya Yamamura and Shu Matsumura as well as all the other members of the Nakashima laboratory for their helpful contribution throughout the course of this work.

Finally, I thank my family for supporting and encouraging me during my student life in Japan.

Vallade Benoit

June 2016

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Artificial intelligence (AI) constitutes an important part of computer science applications. It is widely used in the video games [1] and robotic industries. AIs' performances have also been increasingly publicized; most recently the victory of Google DeepMind's AlphaGo [2] against Lee Se-dol has drawn the world's attention.

In the past, AIs have been designed primarily to address specific tasks, but today's grand challenge is to create general intelligence that can resolve numerous and diverse problems. General artificial intelligence (GAI) is a research area particularly addressing the development of AI which can learn to perform different tasks successfully.

The initial objective of GAI research was to create AI that could successfully perform any intellectual task a human being can. The emphasis is assigned to the AI learning capacity and its adaptability. Beside the desire to reproduce a human mind, another goal is to simplify the AI development process for the industry. Indeed, if an AI is able to learn by itself any task, it could be used to generate a solution to any specific problem. Doing so would automate the AI development process and allow the developers to easily create different AIs.

General video game playing artificial intelligence (GVGPAI) is a branch of the GAI research addressing the development of a controller which can play more than one video game successfully. In GVGPAI, the AI must work out how to play and win games it has never seen before. To do that, the AI is given the game's state and basic information including the applicable actions and the number of players. The advantage of focusing on video games is that it provides a large set of challenging cheap to develop problems.

To better understand the current study a basic understanding of video games' mechanics is required:

The main part of the game's program, deals with all data representing the game's information. These data include information such as the position and size of the enemies, the player's life points and so forth. The game's state corresponds to a snapshot of all these data at a specific iteration of the game.

At every game iteration, the agent must decide on its action. At the minimum, the agent will be given a part or the totality of the current game's state and be told basic information including the applicable actions and the number of players. Then the game will update the game state considering the actions of the other agents in the game and the game physics. Fig. 1 presents the global mechanics of the game loop.

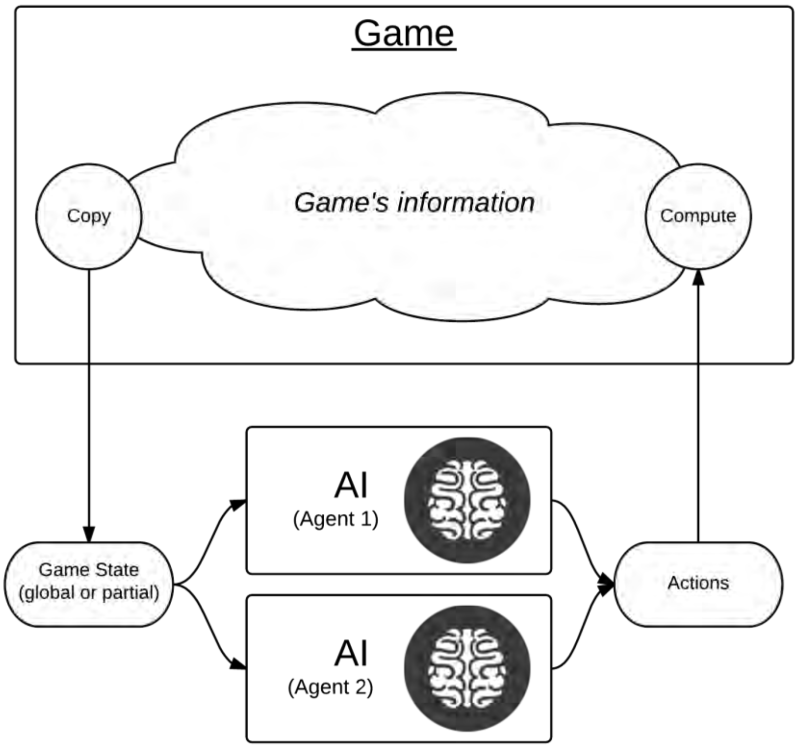


Fig. 1. Game's loop.

Over the year, the proposed solutions have been particularly addressing two ways to address such challenges.

The first one is to design a knowledge-free AI that makes a total abstraction of the game's characteristics. Therefore, the AI is playing the game without being aware of its objectives and game-play. Such solutions are based mostly on tree search algorithms and simulations. A popular planning algorithm for such general AI is the Monte Carlo Tree Search (MCTS) [3][4][5] which produces great results. Its main weaknesses arise when used with complex games with high branching trees and games requiring long-term planning beyond the planner's horizon.

The second way is to make an AI able to learn the characteristics of the game and to use that knowledge to improve itself. The solution proposed in [6][7][8], based on a neuro-evolution of augmenting topology algorithm (NEAT), is a strong candidate in that category. The AI's neural network assimilates the game-play's features and gives back actions depending on the game state. In many games, algorithms of these kinds were found to be superior to planning algorithms. However, these methods tend to discover and to use game loopholes instead of playing the way intended by the game-play.

In our research, we designed a three layers framework which would optimize the adaptability and the efficiency of the AI to all kind of video games.

Our approach suggests a possible solution to these shortcomings by introducing an AI with a more human-like mode of thinking. Generally, humans do not act without a purpose. Their acts are not merely a sequence of uncorrelated actions chosen one at a time, but are chosen as a whole to accomplish a particular strategy.

In this dissertation we suggest a solution to imitate such a process using sub-objectives as a representation of the AI's purpose and separating the strategy and action selection processes.

We detail a design based on a three layer architecture. The first layer is responsible for computing the global strategy of the AI. Strategy comprises a sequence of sub-objectives. The second layer's purpose is to convert this strategy's sub-objectives into a sequence of actions that are directly usable by the agent. Finally, the third layer can override the first and second layers to handle emergencies endangering the agent. In addition we suggest two different implementations for that design and compare their performances.

This dissertation is organized as follows:

The next chapter starts by explaining the main objectives and the motivation of the current research. Then it describes the architecture on which is based the proposed GVGPAI. That description includes all three layers and the global mechanism of the architecture. Afterward, to prove the validity of the proposed solution as well as its ability to adapt to many games, two application examples followed by the results obtained during a competition are presented.

The chapter three presents the sub-objective concept used in our work. This concept is essential to our research and is explained thoroughly in the first parts of the chapter. Then the process used to self-generate the sub-objectives related to a game is explained.

The penultimate chapter details two different AI implementations and their performance evaluation. Each implementation uses different algorithms but is based on the architecture proposed in this dissertation. The second implementation was developed to overcome the flaws observed in the first implementation. The performance evaluation addresses two points. First it checks if the proposed solution has achieved the goals set by our study. Then it determines if that solution is suitable to compete with other GVGPAI solutions.

Finally, we conclude by summarizing the main points of this dissertation.

Chapter 2

Multilayer Architecture for GVGPAI

2.1. Introduction

This chapter provides the foundation on which the rest of our work is built. As introduced, we aim to develop a general video game playing artificial intelligence. The first step in developing an AI is to define its objectives and design.

The design of an AI determines its concept and architecture. While it does not recommend any particular algorithm, it determines the internal mechanism of the AI. The AI's design directly impacts its performance and its ability to achieve the objectives set by our study.

Therefore, the first section details our research objectives and its own distinctive concepts. Then the multilayer architecture on which are based the proposed GVGPAI is described. The third section depicts a few practical examples on which our solution could be applied, thus suggesting its adaptability to various games. Finally we present the results obtained during a single-game based competition using an AI partially based on the current design.

2.2. Research Objectives

This dissertation focuses on the development of GVGPAI able to learn the characteristics of the game. First, this section describes the main targets of the current work. Then it details the motivations behind these targets. And finally, the key concepts on which this work relies are presented.

2.2.1. Main Targets

The main target of this study is to develop a GVGPAI using a more human-like mode of thinking. Generally, humans do not act without a purpose. Their acts are not merely a sequence of uncorrelated actions chosen one at a time, but are chosen as a whole to accomplish a particular strategy.

For example, a human will not make a single step in the direction of a bottle of water because it might increase his cumulative well-being at the end of the day and then wonder again what action he should do next. Instead, a human will first decide to drink because he is currently thirsty and then accomplish a whole sequence of correlated actions leading him to the bottle. This characteristic of behavior makes humans more able to handle diverse situations. Fig.2 depicts the situation described above.

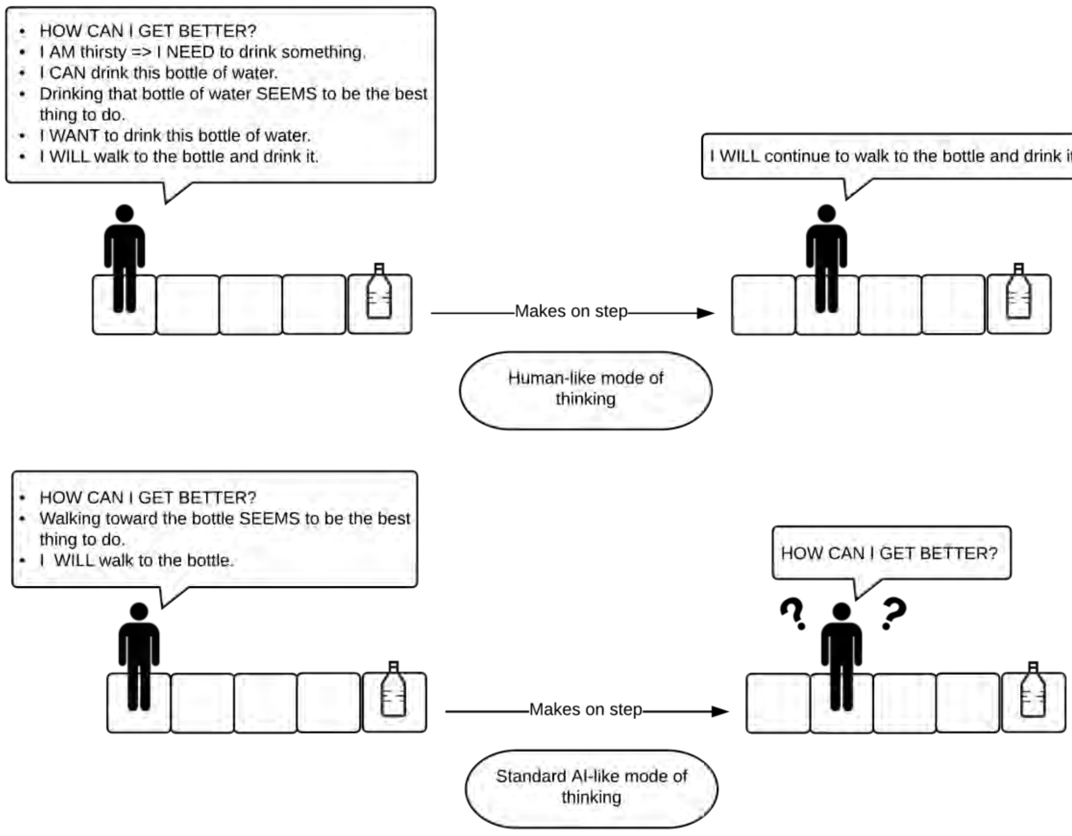

• HOW CAN I GET BETTER?
• I AM thirsty => I NEED to drink something.
• I CAN drink this bottle of water.
• Drinking that bottle of water SEEMS to be the best thing to do.
• I WANT to drink this bottle of water.
• I WILL walk to the bottle and drink it.
I WILL continue to walk to the bottle and drink it.
Makes on step
Human-like mode of thinking
• HOW CAN I GET BETTER?
• Walking toward the bottle SEEMS to be the best thing to do.
• I WILL walk to the bottle.
HOW CAN I GET BETTER?
Makes on step
Standard AI-like mode of thinking

Fig. 2. Difference between human-like and standard AI-like mode of thinking.

When a problem appears, humans can analyze it and ascertain the right set of sub-objectives (what he can do) to solve that particular problem. Then they can elaborate several strategies assembling these sub-objectives in various orders and can choose the most appropriate one to solve the problem. Finally, humans can choose the best actions to accomplish the selected strategy. This process relies on two key points:

- The ability to determine the set of sub-objectives related to a particular problem.
- The ability to separate the strategy and action selection processes.

This study focuses on the development of GVGPAI with such capabilities. First, in order to learn the game's characteristics, the AI will have to be able to discover the sub-objectives related to the game and to assimilate its game-play. These sub-objectives describe what the agent controlled by the AI can do in the game. The AI will be able to assemble them to form various strategies or to transform them into concrete actions. Sub-objectives are a cornerstone of our work and will be thoroughly detailed throughout this dissertation. Then, to separate the strategy and action selection processes, the proposed GVGPAI will be built on a multilayer architecture and each selection process will be assigned to a different layer.

To sum up, the current research addresses the development of GVGPAI able to DISCOVER WHAT it can do in the game, to DECIDE WHAT it want to do and to DECIDE HOW to accomplish it.

2.2.2. Motivations

Using a more human-like mode of thinking should enable the AI to avoid the weaknesses identified on other GVGPAI solutions.

First, this would force the AI to adopt a more consistent and natural way of playing. Discovering what it can do in the game (sub-objectives) and then using it to generate strategies would encourage the AI to use the intended game-play instead of exploiting the game's loopholes.

Secondly, as introduced, the MCTS based methods are weak on games requiring long-term planning beyond the planner's horizon. However a study using a domain-specific algorithm to plan a strategy combined with a MCTS algorithm to plan actions for that particular strategy [9] has shown very satisfying results. This suggests that it might be advantageous for an AI to step back from the game and to elaborate a strategy before to choose its actions.

Furthermore, the sub-objectives as well as the algorithm used to plan the strategy in that study were using domain-specific contents specifically designed to play to the physical traveler salesman problem (PTSP) competition [10]. Therefore, although these results are encouraging, it is essential to develop an AI able to self-generate such domain-specific knowledge. This is what motivates the development of a system able to ascertain the set of sub-objectives related to the game.

2.2.3. Key Concepts

The GVGPAI described in this dissertation are built upon two key concepts: the actions and the sub-objectives.

Every video game is based on the interactions between the agent and the virtual environment described by the game. These interactions are performed using a set of actions made available to the agent. Each one of these actions has a unique effect and is directly executed by the game. The activation of one of these action results in the corresponding modification of the virtual environment. If the agent is controlled by a human player, the actions are triggered by pushing the keys and buttons of the keyboard or game-pad. However, when the agent is controlled by an AI, the selected actions are represented using their identifier within the game.

Sub-objectives are different from actions and cannot be executed nor understood by the game. Sub-objectives represent what the agent can do in the game. They are targets that the agent tries to accomplish using the set of actions provided by the game. Consequently, a single sub-objective can require more than one action in order to be achieved and there may be more than one way to complete it. A sub-objective can be understood as a step of the agent's progression in the game. And a sequence of sub-objectives constitutes a strategy. In general, human players instinctively use sub-objectives. As detailed in the "Main targets" sub-section, humans generally start by selecting a particular sub-objective before to execute the sequence of actions they judge the most fit. The sub-objective implementation used by our study is discussed in more details in the third chapter of this dissertation.

In summary, sub-objectives describe "What" the AI wants to do and the actions describe "How" it wants to do it.

In order to give absolute clarification on the difference between actions and sub-objectives, a concrete example taking place in the context of role playing games (RPG) is provided. In order to progress in a RPG game, the agent may be able to (may want to) do many things. One of those things is to kill a particular monster. This task defines what the agent can do (want to do) and thus is a sub-objective. In order to achieve this sub-objective the agent has several options. Each one is a sequence of actions describing a different way to achieve the same sub-objective. Fig. 3 represents that sub-objective as well as three possible sequences of actions and their consequences. We can observe that even if in all cases the sub-objective is achieved, other parameters of the game take different values depending on the chosen sequence.

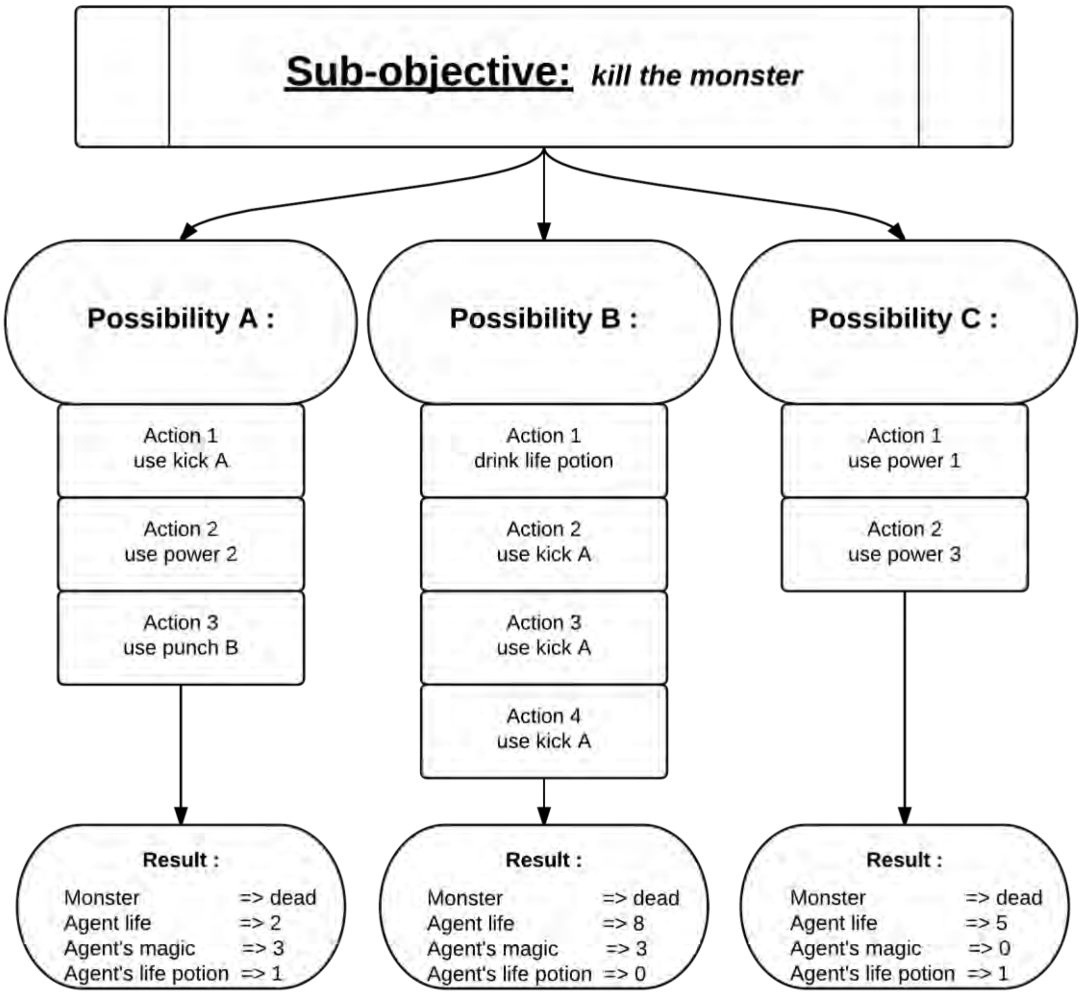

Sub-objective: kill the monster
Possibility A :
Possibility B :
Possibility C :
Action 1 use kick A
Action 2 use power 2
Action 3 use punch B
Action 1 drink life potion
Action 2 use kick A
Action 3 use kick A
Action 4 use kick A
Action 1 use power 1
Action 2 use power 3
Result : Monster => dead, Agent life => 2, Agent's magic => 3, Agent's life potion => 1
Result : Monster => dead, Agent life => 8, Agent's magic => 3, Agent's life potion => 0
Result : Monster => dead, Agent life => 5, Agent's magic => 0, Agent's life potion => 1

Fig. 3. Example of RPG's sub-objective and corresponding potential sequences of actions.

2.3. Multilayer Architecture

The architecture of an artificial intelligence describes its global design. It details how many modules compose the AI, determines their purpose and their way to interact together. However it does not recommend the use of any particular algorithm.

The architecture proposed in this dissertation is designed to be usable on any kind of game and to encourage the AI to use a more human-like mode of thinking. As explained earlier, the two main objectives of this study are to separate the strategy selection process from the action selection process and to enable self-generation of domain-specific knowledge. These objectives are at the core of the current architecture development.

The proposed architecture is divided in three layers which are detailed separately in the following sub-sections. Then the fourth sub-section explains the architecture's global mechanism and the interaction between these layers. Finally the advantages and disadvantages of our solution are summarized.

2.3.1. First Layer

The first layer's purpose is to ascertain the AI's strategy. Strategies represent possible agent's progressions in the game and are composed of sub-objectives. Each sub-objective defines a target which needs to be reached before to start the next one. This layer is divided in two modules which are assigned to a different task.

The first module's main task is to ascertain the sub-objectives related to the current game. Sub-objectives define what can be done in a game and the list of sub-objectives is different for each game. Therefore such knowledge, called domain-specific knowledge, cannot be directly integrated in the code of a GVGPAI. The current solution proposes to self-generate this knowledge using the information provided by the game. Consequently the first module ascertains the list of sub-objectives related to a game using the game states. In addition, the available sub-objectives in a particular situation may not include all the sub-objectives discovered for the game. For example, in the context of RPG games, the AI may have discovered a possible interaction with a particular type of stone. However if the stone is not present in the current situation, the sub-objectives related to that interaction are not available. Therefore, the first module uses a second feature to select the available sub-objectives from the list of sub-objectives discovered until now. This identification process is also performed using the game state.

Then using the available sub-objectives, the second module generates and evaluates multiple strategies. The size of the strategies may be defined by the AI developer. Because the efficiency of a strategy depends on the agent's situation, the evaluation method uses the provided game state to put that strategy in context. Finally, the second module determines which strategy the AI will follow and selects the next sub-objective to be executed. The selected sub-objective is sent to the second layer.

Fig. 4 represents the first layer's operations.

2.3.2. Second Layer

The second layer's objective is to find the best action or sequence of actions to accomplish the sub-objective sent by the first layer. Again, the size of the sequence of actions may be defined by the AI developer. The list of available actions is provided by the game.

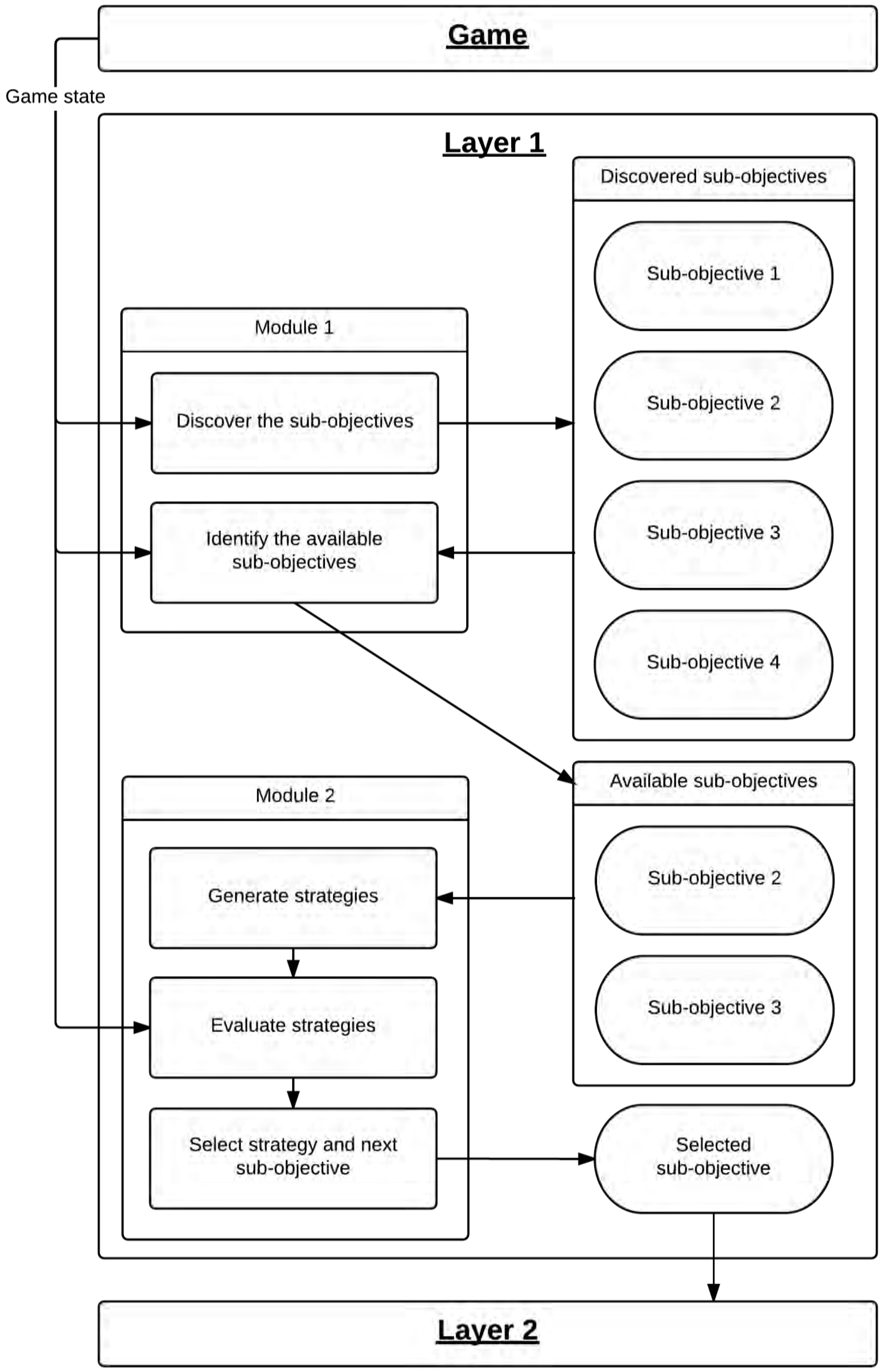


Fig. 4. First layer's operations.

A single sub-objective may be achieved in different ways. And, as for the strategies, the quality of a sequence of actions depends on the agent's situation. Therefore the second layer uses the game state to determine the sequence of actions that is the most likely to complete the sub-objective. Finally, the next action to be executed of the selected sequence is sent to the game.

In the event that the second layer is not able to find a sequence of actions able to accomplish the sub-objective sent by the first layer, the choice of the next action is either performed by selecting another strategy of lesser quality or transferred to the third layer. The selection of which method to use depends on the needs of the AI developer.

Fig. 5 represents the second layer's operations.

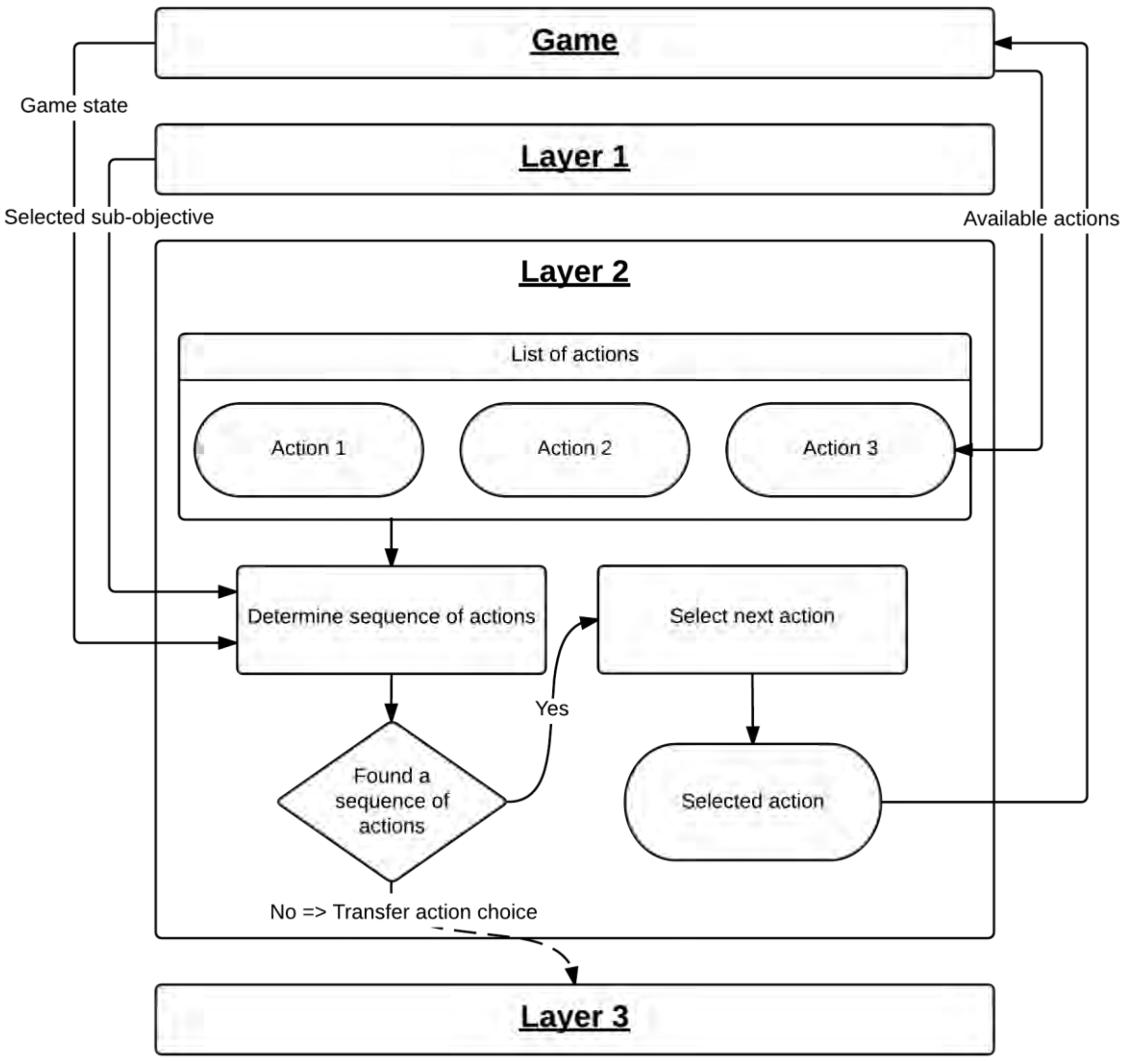


Fig. 5. Second layer's operations.

As a direct consequence of the proposed architecture, the AI becomes more adaptable to the modifications of the agent's situation. Indeed, whenever the game's situation changes, the previously computed sequence of actions might become unsuitable. However, instead of searching for a new strategy, the AI can adapt its sequence of actions to these modifications.

For example, if the sub-objective sent by the first layer is to kill a particular monster, the initial sequence of actions chosen by the second layer could be “use kick A”, “use power 2” and “use punch B”. However if during the fight the enemy makes extra damages (critical hit), thus endangering the agent’s life, some actions allowing the agent to regain health can be inserted in that sequence of actions. By doing so, the initial sequence of actions is modified while the selected sub-objective (kill the monster) remains the same. Maintaining the same strategy is a good way to save processing time and to play in a more natural way. Fig. 6 depicts the situation explained above.

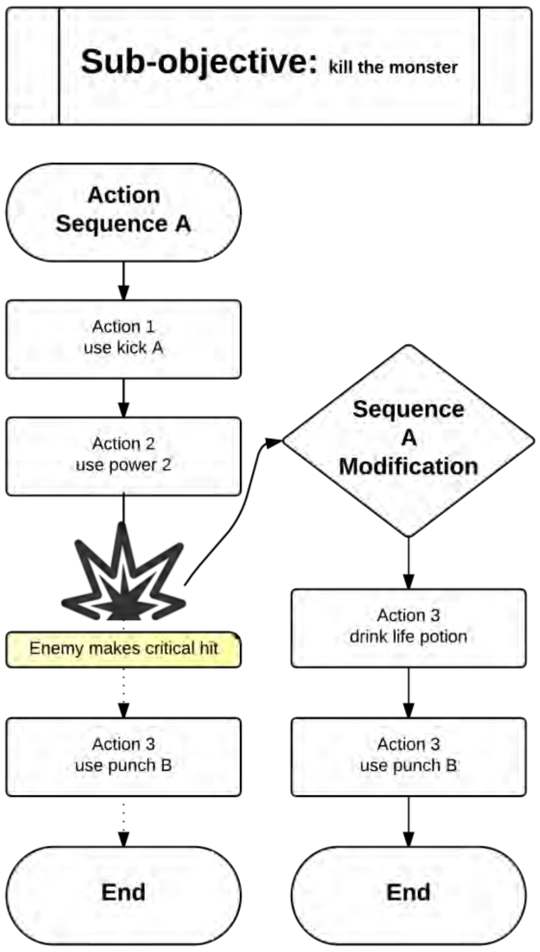


Fig. 6. Adaption of the chosen sequence of actions.

2.3.3. Third Layer

Finally, the third layer handles the emergency cases. An emergency case is an unpredictable event that might definitively prevent the agent to win the game.

As explained above, the second layer can modify its sequence of actions to stick to the game’s situation. However, those modifications are invariably conducted with the goal of accomplishing the current sub-objective. An emergency allows the AI to ignore the current sub-objective and to take drastic measures to save the agent. Therefore, when an emergency case is detected, the third layer overrides the others and generates a sequence of actions leading the agent to a safer situation. For example, in the situation described previously (where the enemy makes extra damages), if the agent runs out of ways to regain health its life becomes critically endangered. From that moment, the third layer is allowed to ignore the current sub-objective and to order the agent to run away from the fight.

The layer 3 is divided in two modules. The first one uses the game state to determine if the agent is currently in an emergency case. If the agent is not endangered, the choice of the next action is transferred to the two first layers. Otherwise, the second module generates the right sequence of actions to lead the agent out of that emergency case. Then it selects the next action to be executed and send it to the game. Another case triggering the second module is when the second layer transfers the choice of the next action because of its inability to find the right solution. In such cases, the third layer computes a single action for the current game iteration.

Fig. 7 represents the third layer's operations.

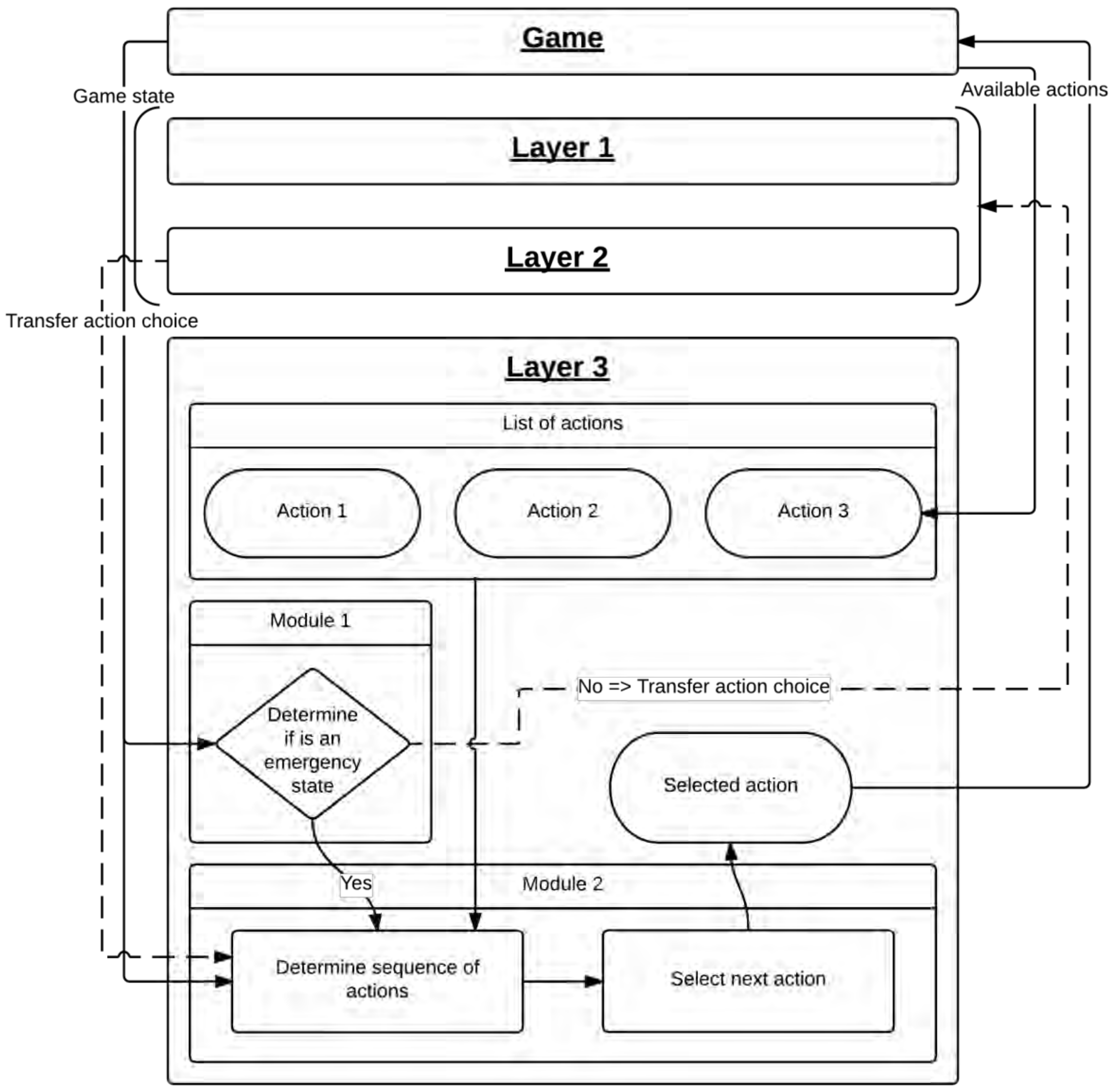


Fig. 7. Third layer's operations.

2.3.4. Global Mechanism

This subsection explains how the three layers and their modules operate and interact together during the game.

At each iteration of the game instance, the AI receives a complete or partial game state which corresponds to a snapshot of the game's information.

The process begins with the third layer's first module analyzing the provided game state to ascertain whether or not the agent is in an emergency case. If the agent is not endangered the control of the AI is transferred to the first layer. Otherwise the third layer keeps the control.

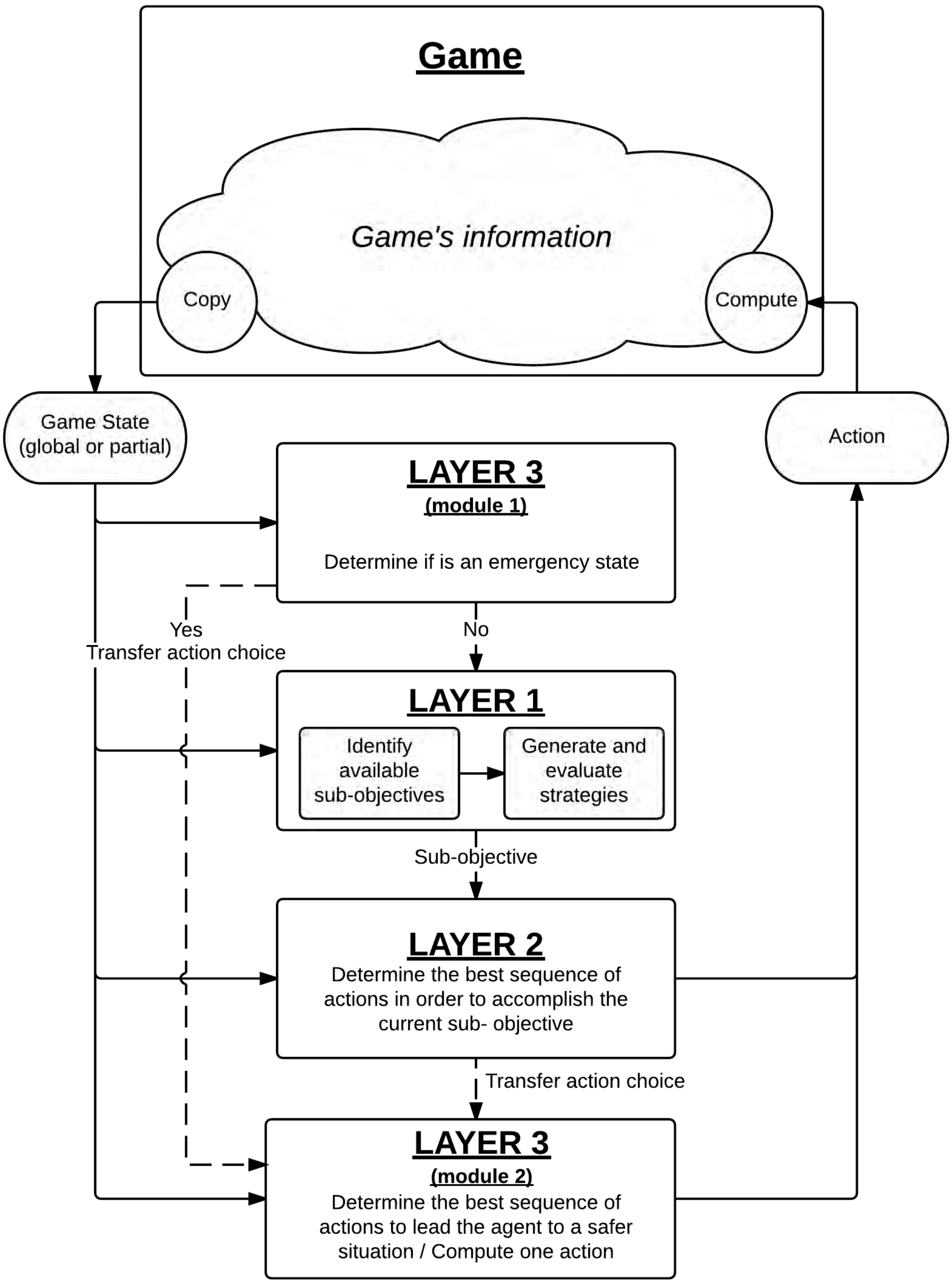
When the first layer receives the control, its first module analyzes the game state to identify the list of sub-objectives available to the agent in the current situation. Because the available sub-objectives may differ depending on the situation, the first module updates that list at each game's iteration. These sub-objectives are picked from a bigger list containing all the sub-objectives discovered until now for that game. They are chosen depending on their relevance with the current agent's situation. Then the first layer's second module uses the list of available sub-objectives and the game state to generate and to evaluate multiple strategies. Finally, it determines which strategy the AI will follow and selects the next sub-objective to be executed. The selected sub-objective is sent to the second layer.

As soon as the first layer's computation is over, the second layer uses the game state to search for the best sequence of actions in order to accomplish the chosen sub-objective. These actions are chosen from a list provided by the game. Finally the next action to be executed is sent back to the game. The latter computes the agent's choice and generates a new game state. If the second layer is not able to find a suitable sequence of actions, the choice of the next action is either performed by selecting another strategy of lesser quality or transferred to the third layer.

When the third layer's second module is triggered, either because its first module has detected an emergency case or because the second layer was not able to find a suitable solution, it overrides the process. In the first case, the second module generates the best sequence of actions to lead the agent to a safer situation and selects the next action to be executed. In the second case, it computes a single action for the current's game iteration. In both cases the selected action is sent to the game.

Finally, when the game instance is over, the AI's learning process is triggered. During the learning process, the AI analyzes all the game's states recorded during the last instance. Using these data, the first layer's first module discovers the sub-objectives related to the game. The list of discovered sub-objectives is updated and will be used during the following game instance. Consequently, during the very first game instance, the list of sub-objectives related to the game is still empty and the AI plays with the sole use of the third layer. Furthermore, because the learning phase is triggered at each end of game instance, the list of sub-objectives related to the game is progressively updated and improved.

In sum, the two first layers can be equated to the AI's logic and the last one is its survival instinct. This division should bring strength to our solution by allowing the use of both strategy and reactivity. Fig. 8 represents the architecture's mechanics as explained above.



Game
Game's information
Copy
Compute
Game State (global or partial)
Action
LAYER 3
(module 1)
Determine if is an emergency state
Yes
Transfer action choice
No
LAYER 1
Identify available sub-objectives
Generate and evaluate strategies
Sub-objective
LAYER 2
Determine the best sequence of actions in order to accomplish the current sub- objective
Transfer action choice
LAYER 3
(module 2)
Determine the best sequence of actions to lead the agent to a safer situation / Compute one action

Fig. 8. Architecture’s mechanism.

2.3.5. Pros and Cons

This last sub-section details the various advantages and disadvantages of the proposed architecture. First of all, here is the list of the advantages:

- The AI separates the strategy and action related decisions:
    - It improves the AI's capacity to adapt its actions to the situation's modifications while keeping the same strategy.
    - It enables the agent to play more consistently (using a seemingly human way of playing).
- The AI self-generates the sub-objectives related to the game:
    - It improves the AI's adaptability to various games.
    - It encourages the AI to use the intended game-play (using a seemingly human way of playing).
- The AI has an emergency layer:
    - It improves the global performance of the AI by handling cases which are not taken care of by the first and second layers.
    - It allows playing on a wider set of games (including games requiring quick reactions).

Then the following list details the different disadvantages:

- The AI self-generates domain-specific knowledge:
    - The AI requires to be trained before to obtain satisfying results.
    - The training phase might be long.
- The AI is based on a complex architecture and mechanism:
    - It is difficult to implement.
    - It may require more computation time than the standard MCTS or NEAT based solutions.

In the main, the proposed solution favors more the human-like mode of thinking and the adaptability than the required computation time.

2.4. Potential Applications

Previously, we illustrated our explanations using the example of role playing games (RPG). This section presents two more application examples based on other types of games. These examples are used to demonstrate our architecture's adaptability to various games. The first example is based on the physical traveler salesman problem (PTSP) and was part of a Computational Intelligence Game (CIG) conference's competition. The second one focuses on Real Time Strategy (RTS) games [11]. RTS games include all the games similar to Starcraft. The latter is actually widely used in the AI research field.

In order to understand how an AI based on the proposed architecture would be adaptable to a game, it is important to identify three of that game's characteristics. Each one of these characteristics is related to a particular layer of the architecture. First we need to determine what are the game's sub-objectives (layer 1). Then, we have to list the set of actions available to the agent (layer 2). Finally, we have to identify what kind of emergency cases the agent may experience (layer 3).

For both examples, the concept of the game is explained and then the corresponding three characteristics are identified.

2.4.1. Physical Traveler Salesman Problem

The physical traveler salesman problem is a game based on the well-known traveler salesman problem (TSP). As a reminder, in the TSP the agent has to visit a list of cities only once each. Its objective is to determinate the best order in which visit the cities while optimizing either the time or the distance. In the PTSP competition, a physical dimension is added to this problem. In order to win the game, a little ship representing the agent has to move within a two dimensional map and to collect as quickly as possible the 10 objective points. The map composed of an infinite number of positions may also contain obstacles and the ship's movements are subject to be altered by the inertia and friction factors. The game stops when the ship has collected all the objective points or if it fails to reach a new objective point in less than 10 seconds. Fig. 9 depicts a map used during the PTSP competition.

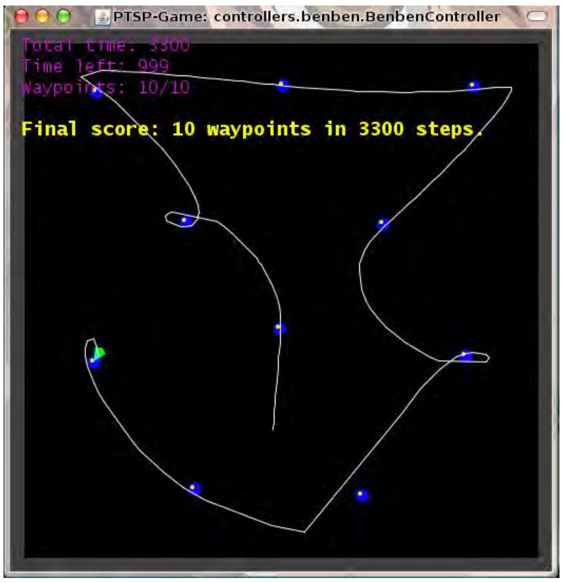

PTSP-Game: controllers.benben.BenbenController
Total time: 3300
Time left: 999
Waypoints: 10/10
Final score: 10 waypoints in 3300 steps.

Fig. 9. Example of PTSP map.

To win the game, the ship has to visit only once as quickly as possible each of the 10 objective points. Therefore, the game's strategy is related to the ship's movements and the order in which it visits these points. Consequently, in this game, the sub-objectives will correspond either to a direction in which the ship should move or to a particular destination which the ship should reach. The strategies would describe a route around the obstacles to reach all the objective points.

The physical equations ruling the game's mechanics and the actions are specified by the competition organizer. The Table 1 details the actions available to the agent.

TABLE I. LIST OF AVAILABLE ACTIONS FOR THE PTSP COMPETITION.

Action ID	**Acceleration**	**Steering**
0	No (0)	No (0)
1	No (0)	Left (-1)
2	No (0)	Right (1)
3	Yes (1)	No (0)
4	Yes (1)	Left (-1)
5	Yes (1)	Right (1)

Finally, the game stops when the ship has visited all the objective points or when it could not reach a new point before the end of the countdown. Consequently, the only emergency case that the agent may experience is to be blocked somewhere against a wall or in a corner and to run out of time. In such case, the agent should stop seeking for the objective points and execute an emergency sequence of movements to unblock the ship.

Based on these observations, we can say that an AI based on the proposed architecture should be able to play to the PTSP competition.

2.4.2. Real Time Strategy Game

In this second application example, we present the real time strategy games without getting into details about any game in particular. Generally, the RTSs' objective is to win by destroying all the enemy players' armies and buildings. For this purpose, the agent will collect various resources and use them to construct buildings or to recruit troops. These troops can be either used to collect more resources or to fight enemies. Some buildings are used to recruit troops whereas others allow purchasing upgrades for either the troops or the buildings and finally others increase the maximum size limit of the agent's army. RTS games can be summarized as a management problem between the resources collected, the army, the building constructed, and the various upgrades. Fig.10 shows a screenshot taken from an RTS game called Starcraft.

Fig. 10. Starcraft screenshot.

In the same manner as above, we determine the three main characteristics of the game. To win the game, the AI has to recruit as quickly as possible a powerful army in order to defend its base and to defeat the enemies. Therefore, the AI has to wisely manage the various facets of the game, the resources collecting, the building construction, the army recruitment, and the upgrades. Each possible modification of these facets corresponds to a sub-objective. For example, “Build more habitations” or “Collect resource A” or “attack the enemy B” are potential sub-objectives.

RTS games generally offer a large number of actions. In addition, the list of available actions changes during the game as the agent creates more troops and buildings. For example, “construct a particular building at a specific position”; “purchase a particular upgrade” or “attack a particular enemy’s building with a particular soldier” are potential actions.

In RTS games the AI generally receives incomplete game states. This can lead to many emergency cases such as an enemy attacking the agent’s base while the agent’s whole army was away to fight another enemy.

As for the PTSP, these observations suggest that an AI based on the proposed architecture should be able to adaptable to RTS games.

2.5. Geometry Friends Competition

This section presents the experimental results obtained during the Geometry Friends Competition organized by the 2014 CIG conference. During this competition we used an artificial intelligence based on the architecture described in this dissertation. However because our research was still at an early stage, the AI did not include all the described features. Nevertheless the results obtained during this first evaluation showed the potential performance of our solution and influenced the direction of our research.

2.5.1. About the Game

The Geometry Friends competition is a game where a rectangle shaped agent and a circle shaped agent have to work alone or together in order to collect a set of objective points. The rectangle agent is able to move on both sides (right and left) and to modify its size (its perimeter length remains identical). The circle agent too can move on both sides and modify its size (its weight changes at the same time) but it can also jump. The agents operate on a two dimensional map (side view) composed of platforms and objective points. The competition is split in three tracks, the cooperative track, the circle track and the rectangular track. We chose to compete in the rectangular track which was the best suited to efficiently evaluate the early version of our solution. The rectangular track only features the rectangular agent. Fig.11 represents the two agents and their actions. Fig.12 depicts a level used in the rectangular track of the Geometry Friends competition held in 2014.

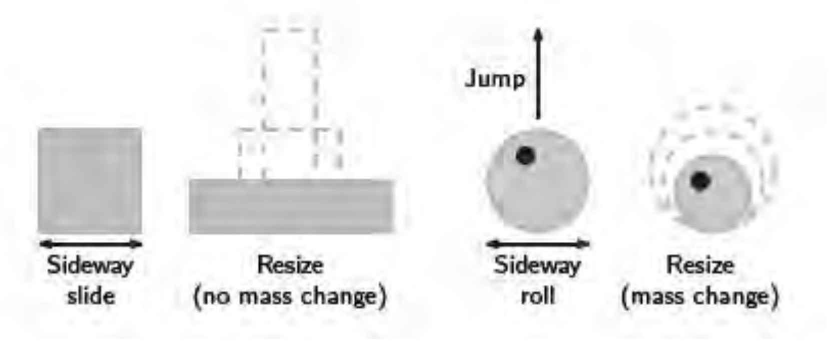


Fig. 11. Geometry Friends Competition's agents and their actions.

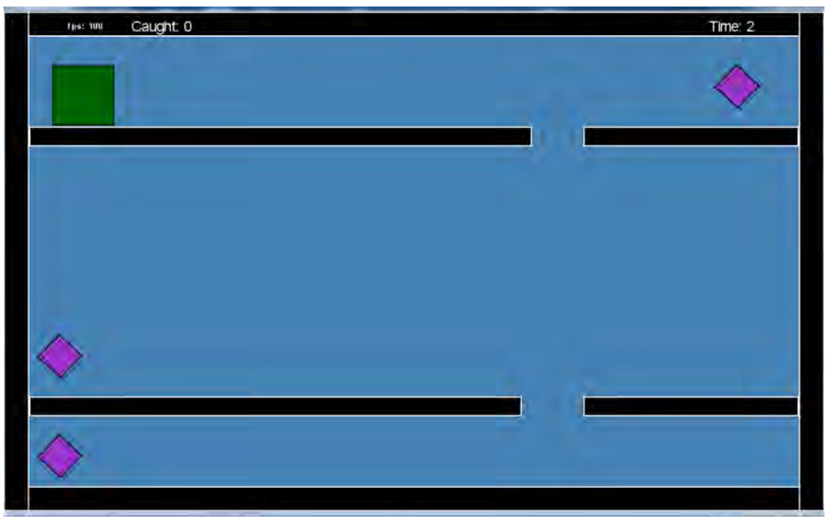


Fig. 12. Level from the rectangular track of the 2014 Geometry Friends competition.

2.5.2. Early Stage AI

We competed using an AI based on the architecture described above. However, this early version of our AI was only composed of the first two layers and could not self-generate domain-specific knowledge yet.

It was developed to be used on the Geometry Friends competition. Therefore the set of sub-objectives were specifically designed for that game. Each sub-objective corresponds to a different objective point, thus a strategy describes the order in which the objective points are visited.

As suggested by the proposed architecture, the first layer searches for the best sequence of sub-objectives. It is composed of an improved version of the particle swarm algorithm [12] generating multiple strategies. A path-finding algorithm based on possibility search [13] is used as fitness function to determine the cost of these strategies.

Then, the second layer is based on a scripted algorithm transforming the sequence of sub-objectives into sequences of actions.

2.5.3. Results and Analyze

We participated to the rectangular track of the Geometry Friends Competition organized by the 2014 CIG conference and we finished second. The competition was conducted on a set of 10 levels composed of 5 public levels (proposed beforehand for training purpose) and 5 unknown levels. AI players were ranked on each level by the total of objective points they gathered and by the time they took to complete the level. To minimize chance each player was tested ten times in each level. The final score of a level was the average of the score of the ten runs. The score of a run was calculated in the following way:

$$ScoreRun = V_{Completed} * \frac{(\max Time - agentTime)}{\max Time} + (V_{Collect} * N_{Collect}) \quad (1)$$

$V_{Collect}$ is the score attributed for each objective collected, $N_{Collect}$ is the total number of objective that were collected, $V_{Completed}$ is a fixed bonus score attributed to the agent if it successfully solves the level (e.g. collects all objectives) and $agentTime$ is the time the agent needed to accomplish the level. Each level will have a time limit ($\max Time$). To avoid promoting over-specialization to the public levels the score on these levels will get a reduction of 50% in the overall score.

The Table 2 details the average score obtained by our early stage AI on each level and the Table 3 details the score obtained by the AI which won the competition (rectangular track). Those tables also describes for each level, the number of run completed, the average number of objective points collected along with the number of objective points present in the level and the average time required to finish the level along with the time limit attributed to that level. A level was considered completed if the agent was able to catch all the objective points in the time limit.

We observe that our AI only completed 60% of its runs, against 69% of completion for the AI who won the first place. However, we can appreciate a global success on the other levels. In addition, our AI performed faster than its opponent on three of the four levels that the AI were both able to complete (level 1, 2, 3 and 4), with a margin of nearly 10 second on the third level.

TABLE II. OUR EARLY STAGE AI'S SCORE.

Level	Runs Completed	Objective Points	Time	Score
1	10	2(2)	12.12(40)	897
2	10	2(2)	8.34(25)	866
3	10	3(3)	23.17(80)	1010
4	10	2(2)	10.79(20)	661
5	0	1(5)	90(90)	100
6	10	3(3)	19.68(40)	808
7	0	2(3)	50(50)	200
8	0	1.8(3)	54.00(60)	180
9	10	3(3)	19.14(35)	753
10	0	0(3)	35(35)	0
			TOTAL SCORE	**5475**

TABLE III. FIRST PLACE AI'S SCORE.

Level	Runs Completed	Objective Points	Time	Score
1	10	2(2)	12.46(40)	889
2	10	2(2)	10.05(25)	798
3	9	2.9(3)	32.83(80)	880
4	10	2(2)	9.06(20)	747
5	10	5(5)	41.64(90)	1037
6	0	1(3)	40(40)	100
7	10	3(3)	20.93(50)	881
8	10	3(3)	21.95(60)	934
9	0	2(3)	35(35)	200
10	0	0(3)	35(35)	0
			TOTAL SCORE	**6466**

Despite the significant score difference with the winner of the competition, the overall performance of our early stage AI was satisfying. The AI was able to efficiently play and win the game and was even faster on some levels. This indicates that our AI was either using a better action selection or that it required a fewer computational cost. Furthermore, it is possible that the absence of the third layer played a role in the loss of performance obtained on levels 5, 7, 8 and 10.

Not having access to the opponent's AI source code or to competition's videos, we could not determine more precisely the reasons of the successes and failures of our AI. However, even if without more details those results are not enough to conclude on the real capacity of our solution, they support its validity and show its potential.

2.6. Summary

In this chapter, we proposed a multilayer architecture which respects the two key points of this study. First the separation of the tasks over multiple layer forces the separation of the strategy and action selection processes. Then the proposed architecture provides for a domain-specific knowledge generating feature (in the first module of the first layer). By doing so, it should enable a more human-like mode of playing. These choices were taken to overcome the weaknesses of the other GVGPAI solutions.

The described architecture is based on three layers and a different task is assigned to each one. The first layer ascertains the sub-objective related to the game and uses them to generate multiple strategies and to select a sub-objective. The second layer searches for the best sequence of actions in order to achieve that sub-objective. Finally the third layer is responsible to handles emergency situations.

An AI partially based on the proposed architecture was developed to compete on the Geometry Friends competition and obtained satisfying results. Because this AI was developed to compete on a single game, the obtained results are not conclusive. However they are encouraging enough to warrant further research on the proposed idea.

The next step is to determine how to implement and ascertain those sub-objectives. Sub-objectives are at the heart of this study and are critical for the development of the proposed GVGPAI. Therefore, the following chapter explains thoroughly the discovery process and representation of these sub-objectives.

Chapter 3

Sub-objectives Discovery

3.1. Introduction

This chapter details how to ascertain the set of sub-objectives related to a particular game. As explained in the previous chapter, our general artificial intelligence plans actions to achieve the strategies it establishes, and each strategy comprises a sequence of sub-objectives.

These sub-objectives can be interpreted as steps of the AI's progression in the game. And because each game's content and game-play is different, the available sub-objectives are different too. For example, while in Pacman one sub-objective could be to reach a particular position, in Invader it could be to destroy a particular enemy. Therefore in order to be able to play to every kind of games our general artificial intelligence must comply with the following requirements:

- The sub-objective's representation must be standardized so that it can represent every possible sub-objective.
- The AI must be able to automatically ascertain the set of sub-objectives related to a game.

In our work, sub-objective's representation is highly related to the game data representation. Hence this chapter starts by introducing the game data representation. Secondly, the concept of sub-objectives and their representation is detailed. Then the extraction procedure is explained thoroughly. Finally the detection results obtained by the AI on a set of games are presented to validate the proposed procedure.

3.2. Game Data

As previously stated, the game data represent all the information about the game. The game data representation is the way in which these data are formatted. And a game state is a snapshot of these data's values at a precise iteration of the game. The game states are entirely or partially sent to the AI so that it can choose its next action.

In theory, the way to represent the game's information is up to the game developer. For example, in the game Pacman, the game's information would likely include the position of the phantoms. However, that same information could be represented differently. It could be composed of couples of integer or even real numbers representing the phantom's coordinates on the map. Or it could also be represented by a screenshot of the game.

In practice, GVGPAI intelligence studies define their own standard representation so all the games the AI may have to play will format their information in a similar way. The current section describes the game data representation used in the proposed solution.

3.2.1. Game Data Representation

This sub-section details the format chosen for the game data and explains its underlying concept.

The game data represents all the information about the game. Most of this information refers to the game's content. This content can be represented as a set of objects, e.g., the characters, the doors, the items, each one constituting a separate object of the game.

These objects will be formatted separately in a similar way. Each object will be linked to its own unique identifier (id), which includes the object type and its number. The number identifies each object, whereas the type enables them to be pooled by family (e.g., all the doors). Along with its id, each object is linked to the set of information representing its characteristics. For example, a door would be linked to its position and its state (open, closed, or locked). A character would be linked to its position but also to its life points, its inventory contents, and all other information necessary to define it completely.

Nevertheless, some information cannot be linked to any object of the game. For example, the time does not define any object in particular. To fit this information in the same way as the others, we create an artificial object called the game statistic. This object will include all neglected data and will be formatted in the same way as the regular objects. Fig.13 depicts the format used for a single object as detailed above.

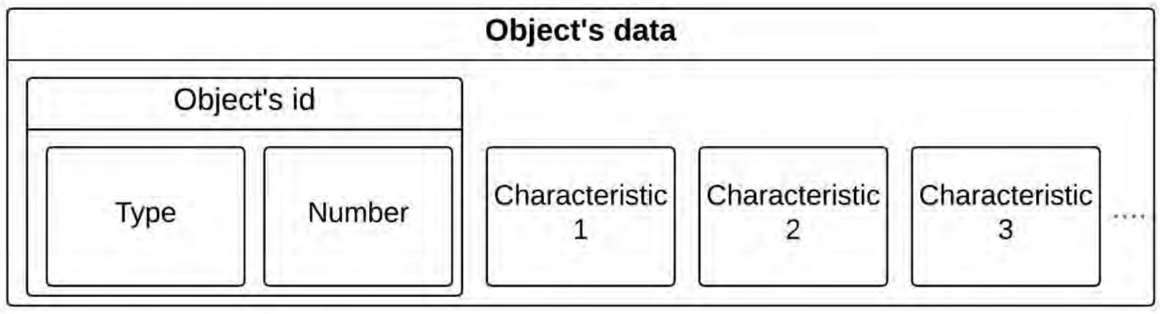


Fig. 13. Object's data format.

The complete game data representation is the composition of every formatted object including the game statistic. Using this format the game data representation should be able to include every piece of information about the game. The Fig.14 represents a game and its corresponding game data formatted using the proposed game data representation.

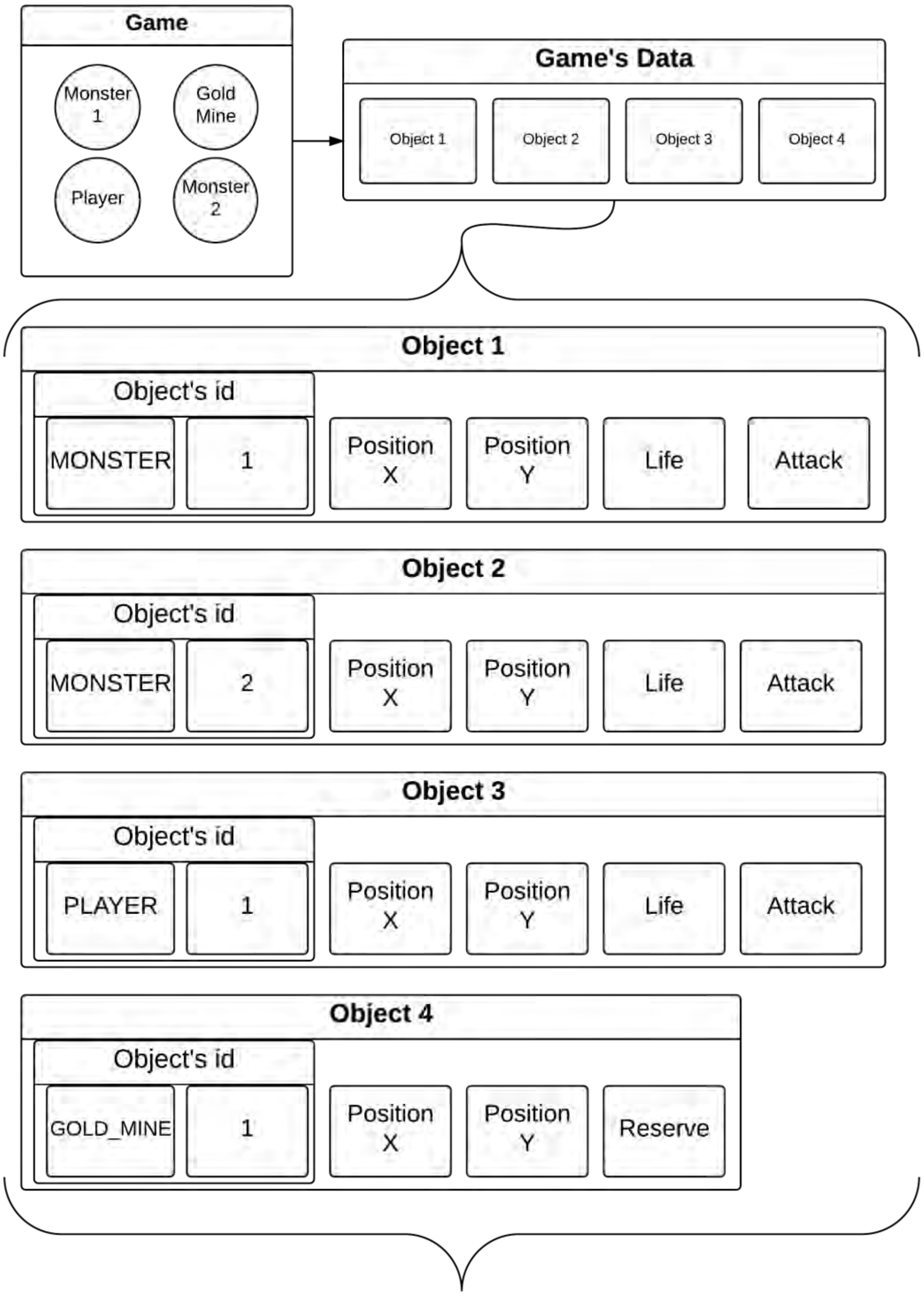


Fig. 14. Example of game data.

The chosen game data representation proposed in this dissertation standardizes the format used by all the games the AI will play. However, the types (float, int, string, bool) used by the data in this representation are not standardized and may change depending on the game developer choices. This particularity raises a technical issue whose solution is explained in the next sub-section.

3.2.2. Multi_Data Class

The following sub-section details some technical aspects necessary for a better understanding of the proposed solution. Programming languages are either dynamically or statically typed. This describes how types (Float, Integer, Boolean …) are bounded to variables. We developed our solution using C# which is for the most part a statically typed language where variables are bound to a type during the compilation of the program.

The principal consequence is that the signatures of the functions used by the AI need to be defined for sets of variables with specific types. Because the current AI is developed to be adaptable to many games it may receive game states using different sets of types.

The class Multi_Data was designed to address that problem. As shown in the Fig.15, Multi_Data objects are used to wrap the data provided by the game and to "transport" it within the AI program until it reaches the right function. There it can be used as such or unwrapped to be treated.

The Multi_Data class's main features are:

- Wrap and Unwrap data :
  - Each Multi_Data object can be used to wrap a single variable (ensemble of variables needs to be wrapped separately).
  - A Multi_Data object conserves the value and the type of the original variable at all time. But can be used to transfer this datum without impacting the AI functions' signatures.
  - The Multi_Data can then be unwrapped into its original type or any compatible type (using C# implicit conversion system).
- Classify data :
  - During the wrapping process a category is attributed to the variable.
  - The chosen category depends on the variable original type and can be NUMBER (for float, double, int …), LOGICAL (for bool), LABEL (for char and string).
  - The category doesn't impact the AI functions' signatures and can be used to simplify the treatment of the data within the AI.
- Compare data :
  - The Multi_Data class provides methods to compare the variable wrapped in it without unwrapping it.

In the proposed solution, all the data provided by the game are wrapped into Multi_Data objects before entering the AI.

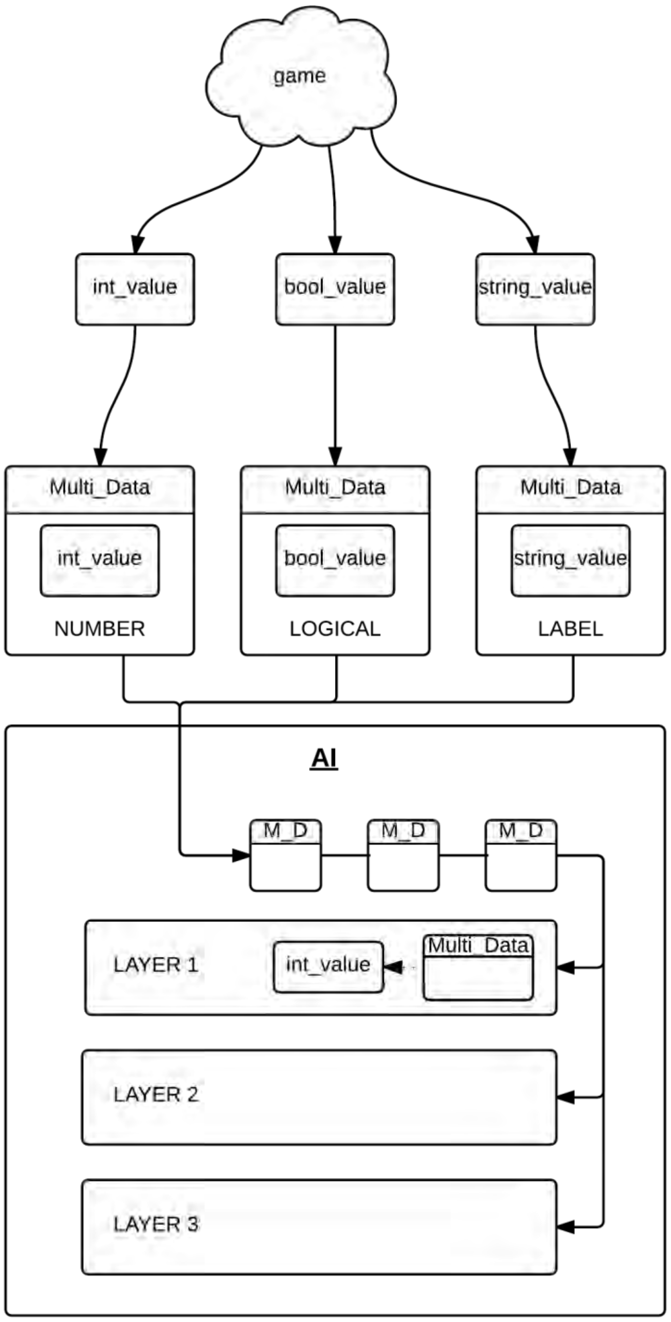


Fig. 15. Utilization of Multi_Data Objects

3.2.3. Game State Generation

A game state is a snapshot of the game data's values at a particular moment of the game. These snapshots are used by the AI to determine its actions. The game state generation is generally composed of three steps. During the first step a snapshot of the data's values is taken and these values are formatted using the game data representation defined previously. In the second step the game state can be partially modified. This step is optional and the modifications (information removal, noise introduction etc) are chosen by the developer of the AI depending on his purpose. Finally all the values are inserted in Multi_Data objects. At the end of the generation process, the data contained in the game state are usable by the AI.

3.3. Sub-objectives

Sub-objectives are the unitary blocks composing the AI's global strategy. This section explains the concept on which is based these sub-objectives and describes their representation in the AI's program.

3.3.1. Concept

In real life, a strategy is a set of guidelines leading toward a specific objective. Each guideline defines a target which needs to be reached before to start the next one. However guidelines do not indicate how to accomplish that target. For example, in order to find a job, ones needs to first buy a suit, write a resume, do interviews and finally to sign the contract. In this case, each target (buy a suit, write a resume, do interviews, sign the contract) describes WHAT the person can do to get a job but does not explain HOW to achieve these targets.

In our researches we refer to these guidelines as sub-objectives. Consequently, sub-objectives describe WHAT is possible to do in a game. In other words, it defines in WHAT ways the AI can interact with the game. Our concept of sub-objective is defined by the following rules:

- Each sub-objective describes a unique interaction.
- All possible interactions provided by the game are defined by its set of sub-objectives.
- Achieving a sub-objective may require one or more actions.
- There may be more than one way to achieve a sub-objective.

3.3.2. Sub-objective Representation

In order to represent such sub-objectives in our program, their representation must comply with this additional set of rules:

- Sub-objective representation must be standardized (over all the games).
- Sub-objective representation must be able to represent any kind of sub-objectives.
- The AI must be able to ascertain the set of sub-objectives related to a game.

As explained previously, the game is entirely defined by the game's information. And this information is entirely represented by the game data. Therefore interacting with the game amounts to interacting with its data. Interacting with a game datum is equivalent to modifying its value. For example, a possible sub-objective may be "kill that enemy". This is equivalent to asking to modify the value of the datum corresponding to that enemy's life to zero. However it does not describe how to achieve this sub-objective and the player is free to use any way he want to kill the enemy. Because all the game information is represented by the game data; every possible interactions of a game can be represented as the modification of one of its value.

Consequently a sub-objective consists of a particular game datum and a modification order. The game data provided to the AI are formatted using the game data representation. Therefore our sub-objective representation is related to the game data representation. Firstly, the datum we wish to modify is represented by the id of the object containing it and by its index within that object's characteristics. This index (as defined by the object's data format described in Fig.11). Then the modification orders are split in two categories, the directions and the targets. A direction is an order contained in the following list:

- Equal (equal to the target / all data types)
- Different (different from the target / all data types)
- Increase (increases the value / only for numbers)
- Decrease (decreases the value / only for numbers)

When the chosen direction is "equal" or "different" a modification order of type target is added. The target takes a particular value which the datum must equal or differ with. Fig.16 represents the sub-objective representation.

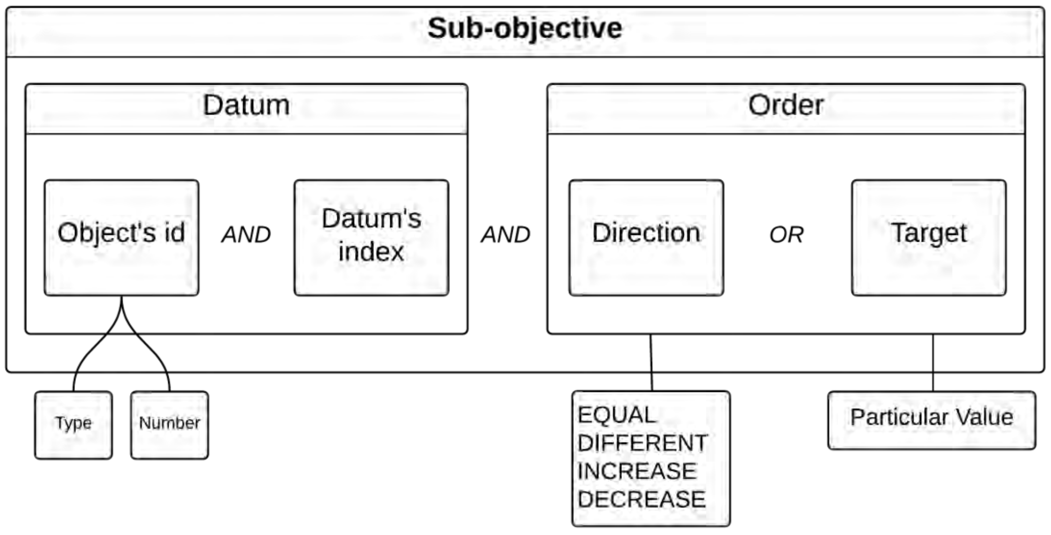


Fig. 16. Sub-objective representation.

3.3.3. Sub-objective Pattern

All the possible sub-objectives of a game can be represented as the modification of one of its datum. However, because there is a different sub-objective for each possible modification, game can be related to sizable the set of sub-objectives. Therefore it is important to be able to group these sub-objectives by family.

We refer to a family of sub-objectives as a sub-objective pattern. A sub-objective pattern regroups all the sub-objectives which are modifying a particular datum of a particular type of object. This means that for each type of object of the game a different sub-objective pattern is generated for each datum linked to that type of object. And that each sub-objective pattern records all the possible modifications for its datum.

The following Fig.17 depicts the sub-objective pattern representation.

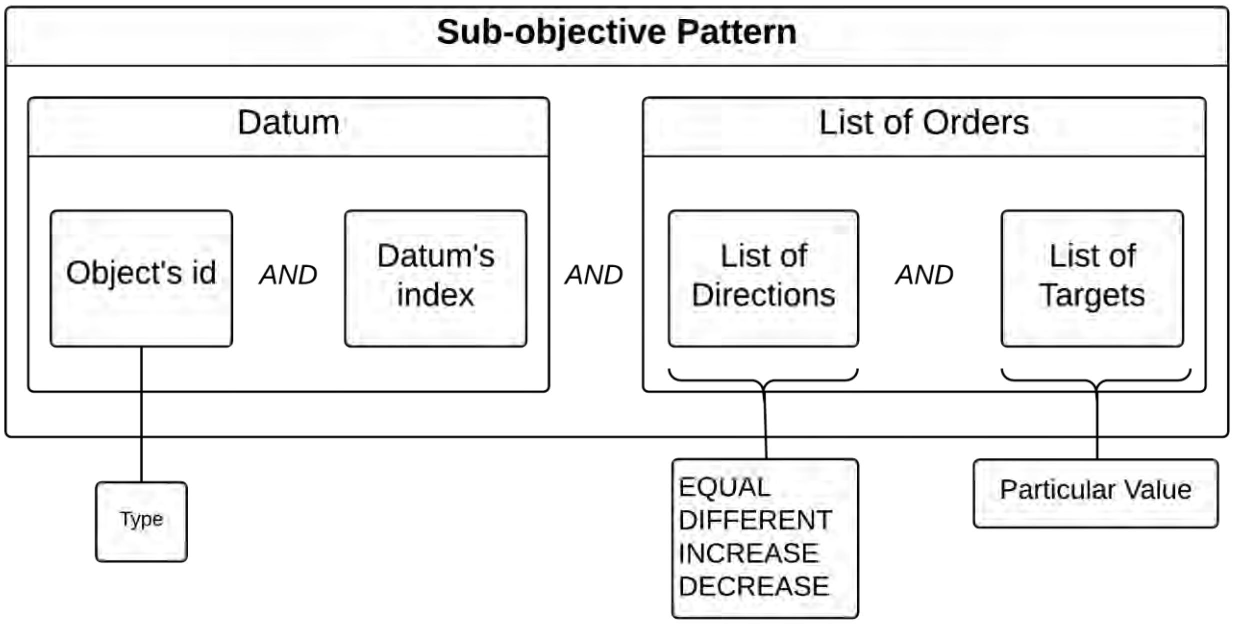


Fig. 17. Sub-objective pattern representation.

The use of these patterns is detailed in the following section about the sub-objective discovery and identification processes.

3.4. Sub-objective Discovery and Identification

The AI must be able to automatically ascertain the set of sub-objectives related to a game. This operation is actually divided in two processes.

At first, the AI has no knowledge of the game and it must discover every possible sub-objective. The discovered sub-objectives are then recorded so they can be used to generate strategies. This phase is called the discovery process.

However all the sub-objectives are not available at all times. For example, the AI might have discovered a way to interact with the object "ball" of the game. But if there is no "ball" in the current instance of that game, the corresponding interactions are not available. Therefore, in order to generate consistent strategies, it is important to reduce the list of sub-objectives to those available at that time. This phase is called the identification process.

The discovery and identification processes are both parts of the architecture's first layer's first module, layer, but these processes are used at different moment of the game. The identification process is used at each game iteration while the discovery process is used at each end of game instance.

3.4.1. Discovery Process

The purpose of the discovery process is to discover all the sub-objective related to the game. This process uses the game state provided by the game. All the game states sent during an instance of the game are recorded. Then, at the end of that instance, the discovery process is performed by analyzing the recorded game's states. The process is divided in three steps.

During the first step, all the potential sub-objective patterns are generated. As explained in the previous section, a sub-objective pattern defines a family of sub-objectives related to a particular datum within a particular type of object of the game. In addition each sub-objective pattern will record all the possible modifications for its datum.

In the first step, the type of the objects used during the current game instance is analyzed. If a type of object which has not been encountered yet is detected, a new sub-objective pattern is generated for each datum related to that type.

At this point of the process, the new sub-objectives patterns are linked to their type of object and datum but their list of possible modifications is left empty. The sub-objective patterns generated during this step are added to the AI's sub-objective patterns list.

Fig.18 depicts the sub-objectives patterns generation based on an example of game state.

The second step's purpose is to ascertain the list of possible modifications of each sub-objective pattern. The game states provided by the current game instance are analyzed in such a way that the evolution of each datum is evaluated through time. For each datum, the possible modifications (direction and targets order) are selected depending on the evolution of that datum values. Finally the list of directions and targets related to a datum are recorded in the corresponding sub-objective pattern.

For example, if the value of a datum is increasing during the game, then the direction "increase" will be added to the modification list of the sub-objective pattern linked to that datum. Again a new target will be generated for each different value taken by the datum.

Fig.19 shows the generation of the list of directions and targets corresponding to the previously generated sub-objective pattern using the given game states.

The third step determines the relevance of the generated sub-objectives patterns. As explained earlier, the set of sub-objectives related to a game can be quite sizable. Therefore it is important remove all the irrelevant sub-objective patterns.

When the value of a particular datum never changes or always evolves in the same manner (based on a pattern or an affine function), then this datum is independent. Therefore, the AI will never be able to modify this datum voluntarily. For example, the datum corresponding to the time should be regarded as independent. The sub-objectives pattern related to independent data will be regarded as irrelevant and moved to an irrelevant sub-objective patterns list.

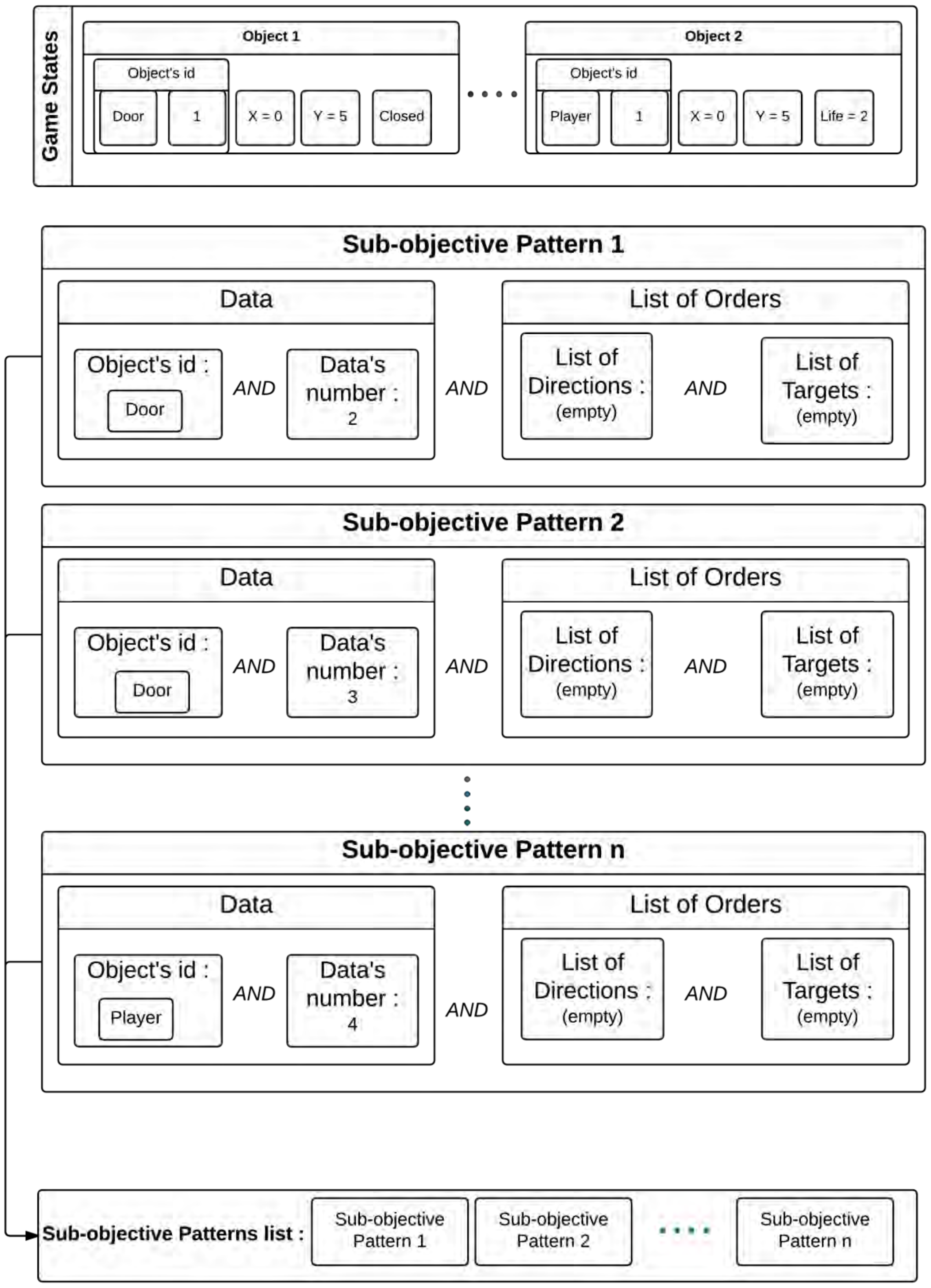


Fig. 18. First step of the sub-objective discovery process.

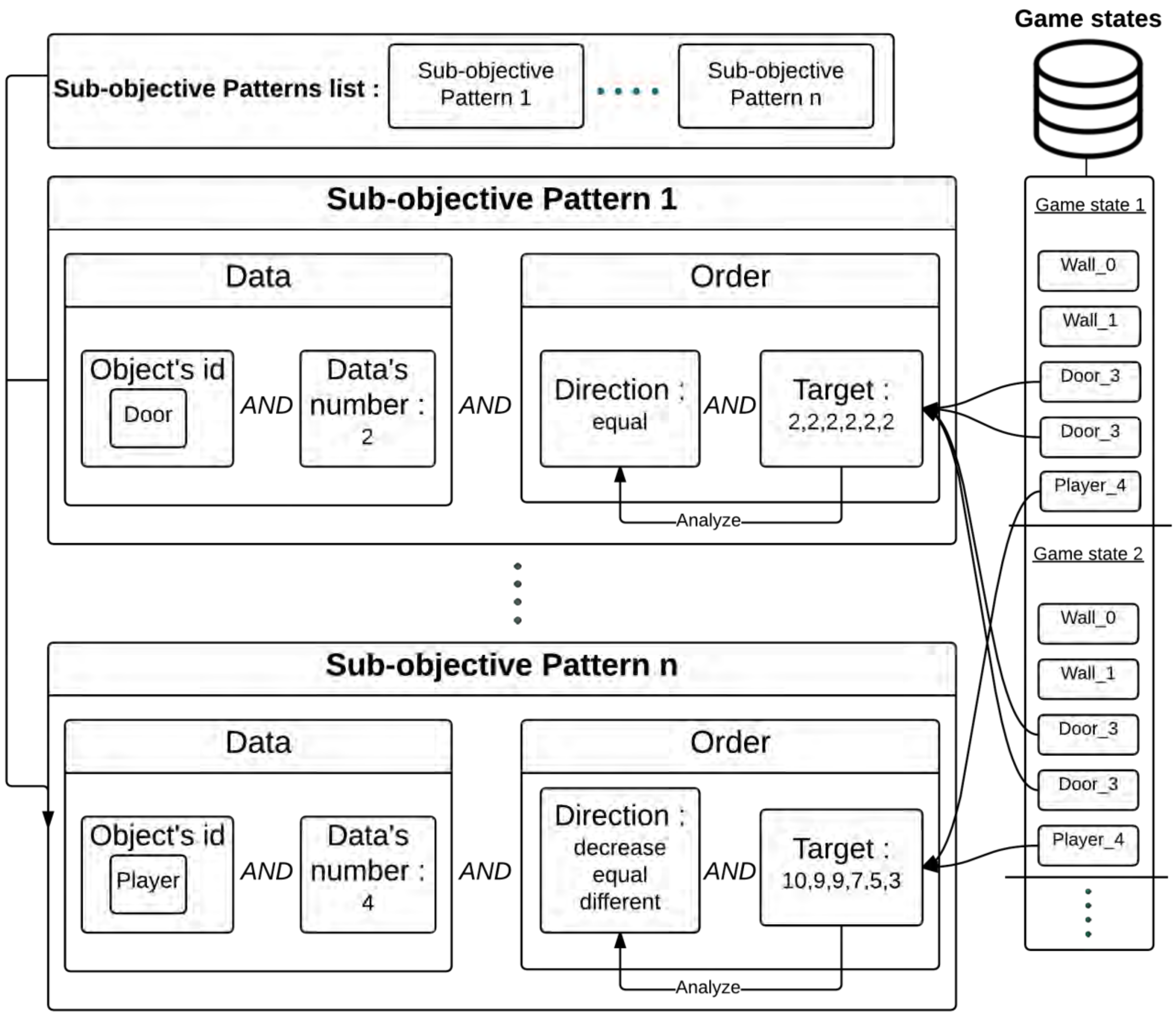


Fig. 19. Second step of the sub-objective discovery process.

When, in the following game instances, a datum previously regarded as independent is found to evolve in a different way, the corresponding sub-objective pattern is moved back to the sub-objective patterns list.

Fig.20 presents an example of exemption of sub-objective pattern.

At the end of the discovery process, we obtain two lists of sub-objective patterns. One filled with irrelevant sub-objective patterns and the other one filled with usable sub-objective patterns describing the modifiable game data and the possible modifications (direction and target orders).

This process is repeated the end of each new instance of the game and the list of sub-objectives is updated.

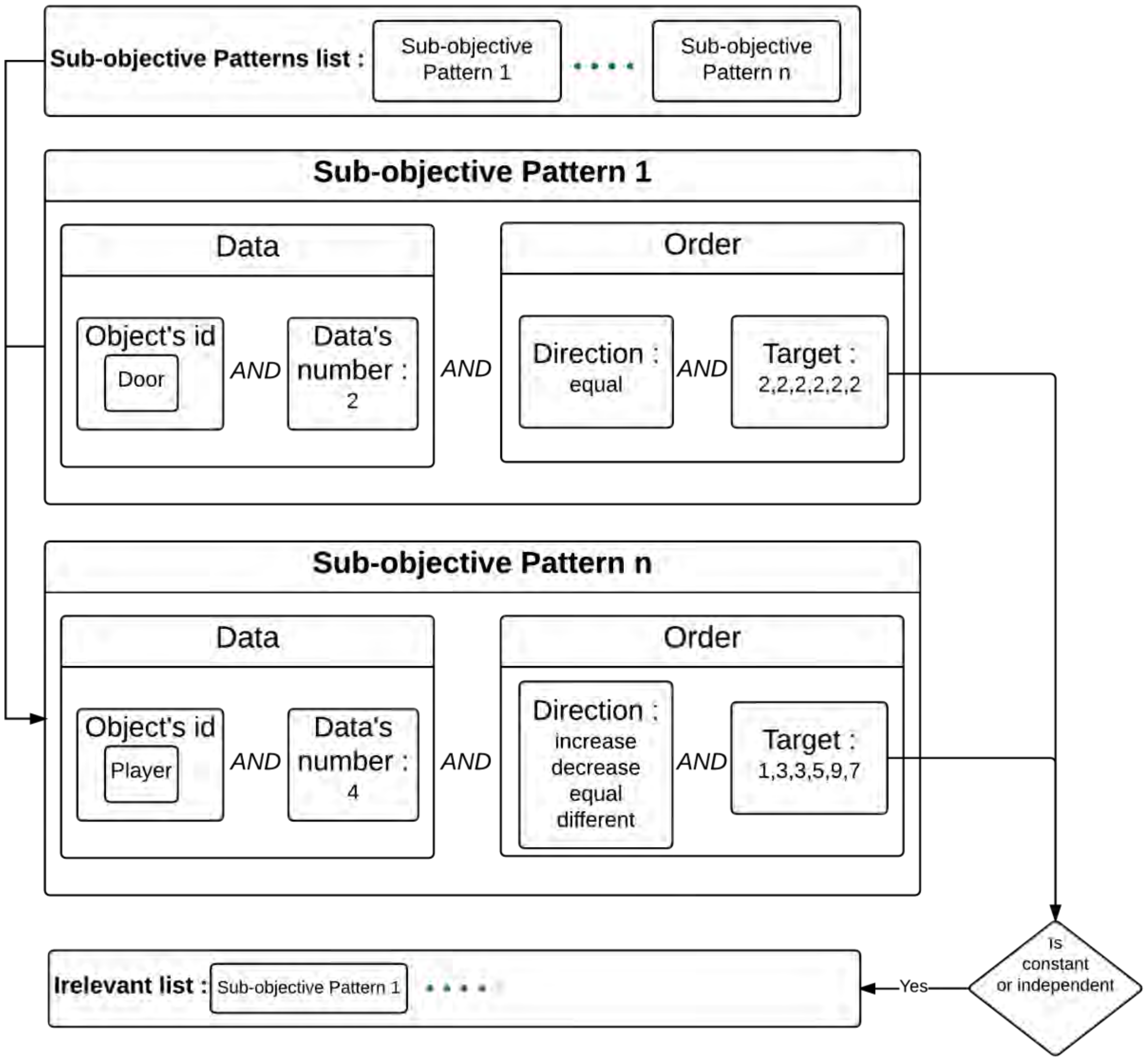


Fig. 20. Third step of the sub-objective discovery process.

3.4.2. Identification Process

The purpose of the identification process is to determine the list of sub-objectives which are usable in the current iteration of the game. This process uses the game state provided by the game and the list of sub-objective patterns generated by the discovery process. It is performed each game iteration before the strategies generation process.

During this process, the provided game state is analyzed and all the objects present in the current game iteration are listed. For each object in that list, the object's type is compared with the type of the sub-objective patterns contained in the list previously generated by the discovery process. When a match is found the set of sub-objectives corresponding to this sub-objective pattern is generated.

This means that a different sub-objective is created for each modification order contained in that sub-objective pattern's order list. In addition, all the generated sub-objectives will have the same type than that pattern and the same number than the object which is currently being compared. After all the sub-objectives usable during the current iteration have been identified, the AI can start generating strategies composed of these sub-objectives.

Fig.21 depicts the sub-objectives generation based on an example of game state and the list of sub-objective patterns introduced in the previous figures.

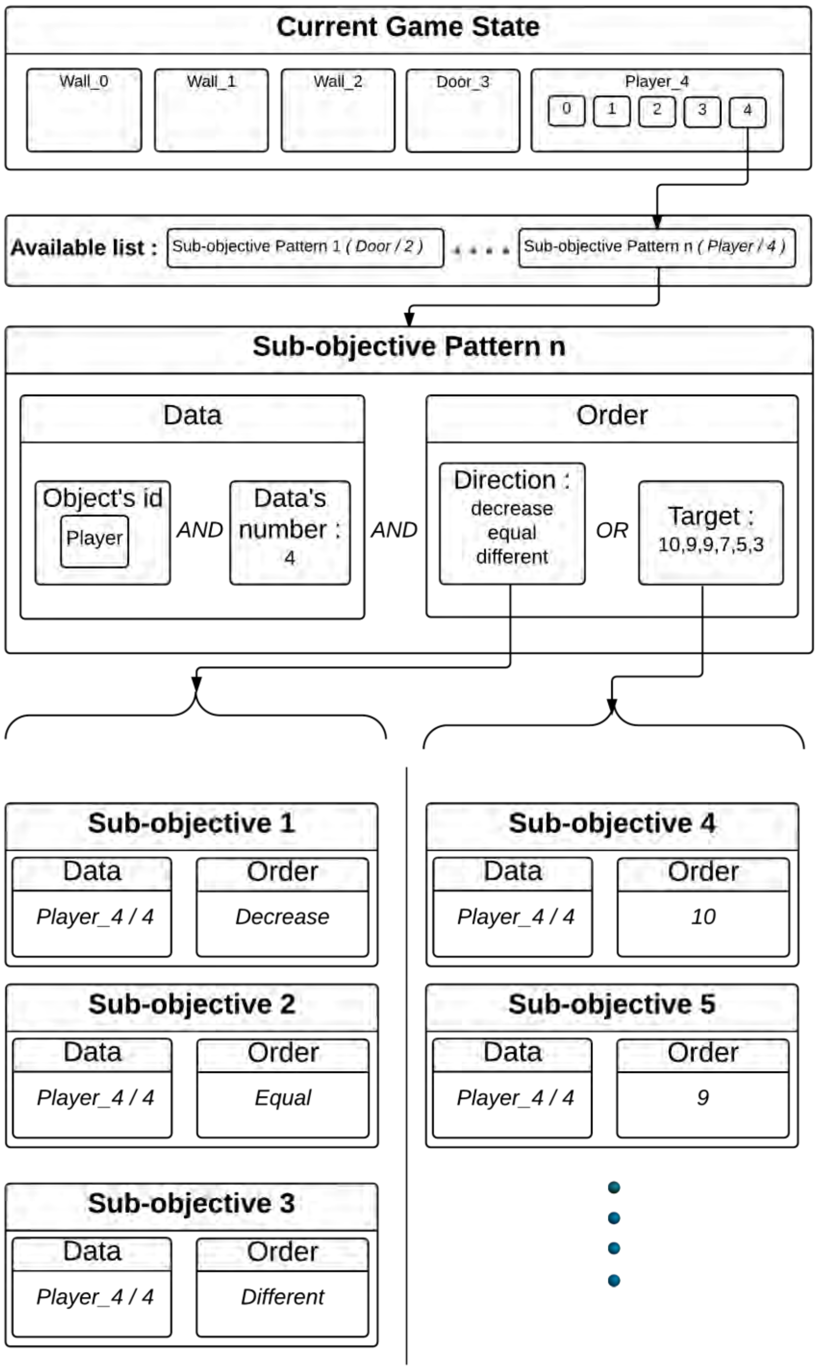


Fig. 21. First Sub-objective Identification Process

3.5. Performance Evaluation

This section presents some experimental results obtained during the evaluation of our proposal's performance. During those experiments the ability of the proposed solution to ascertain the right set of sub-objectives is assessed. This ability is crucial because it directly impacts our AI's ability to adapt to various games.

The discovery and identification processes are integrated in the first module of the first layer detailed in the previous chapter. These methods are directly implemented in the AI's program and are performed automatically at the end of each iteration (for the identification process) and the end of each game instance (for the discovery process). In order to be reusable during the next game instances, the results obtained by the discovery process should be recorded (using for example xml files).

We set up two experiments, each one based on a different game. These games are using a totally different game play; the first one is a card game, and the second one is a motif game.

First, the game-play of the two games is explained. Then the results are presented and analyzed.

3.5.1. Games

To prove the adaptability of our AI to various games and game-plays we created two different games:

- Card Game : The first experiment is based on a card game. This game is using a deck of 10 cards. Each card is associated to a number positive or negative of gain points and of life points. The game runs during 10 iterations. Each iteration, 3 cards are randomly drawn from the deck. Then, knowing the cards' effects (gain and life points), the player must choose one of them. The player possesses itself a number of gain points and life points. At each turn, these two characteristics are affected depending on the chosen card's effects. The game stops when the player has played 10 turns or if its life points reach zero. The player's objective is to maximize its gain points while staying alive. Its score corresponds to its number of gain points at the end of the game. however if the player dies, its score is set to 0. The cards are generated so the two effects (gain and life points) are from opposite sign. Therefore a card with a positive number of gain points will have a negative number of life points and vice versa. Furthermore the cards are carefully generated so there is the same number of cards of the two types in the deck.
- Motif Game : The second game is based on a motif reproduction problem. In this game, the game and the player both own a square board composed of 3 by 3 cells. Each cell can be either activated or not. When the game starts, 5 cells are randomly activated on the game's board and all the cells of the player's board are inactivated. The game stops after 5 turns. Each game's iteration the player has to choose one cell which will change of state. The player's objective is to reproduce the game's board and its score correspond to the number of cells in the right state.

3.5.2. Results and Analyze

During this experiment, we assessed the ability of our AI to ascertain the right set of sub-objectives related to the game. This ability is one of the key points of the proposed GVGPAI. It will determine if the proposed solution is adaptable to many games and if one of the objective set by this study has been reached.

For this purpose we extracted the list of sub-objectives generated by the AI on the two games presented previously. Table 4 presents these sub-objectives in an understandable form. The sub-objectives are split in two groups depending on their actual relevancy in the game.

TABLE IV. ASCERTAINED SUB-OBJECTIVES

	Card Game	**Motif Game**
Interesting sub-objectives	Increase player's gain points Decrease player's gain points Increase player's life points Decrease player's life points	Modify Cell 1's State Modify Cell 2's State Modify Cell 3's State Modify Cell 4's State Modify Cell 5's State Modify Cell 6's State Modify Cell 7's State Modify Cell 8's State Modify Cell 9's State
Irrelevant sub-objectives	Increase card's gain points Decrease card's gain points Increase card's life points Decrease card's life points	

The results show that the proposed AI has been able to efficiently ascertain the list of sub-objectives related to the game. Although some irrelevant sub-objectives have been found in the first game, all the important features of the game-play were ascertained. This suggests that the proposed sub-objective representation and discovery process is working and that our AI is adaptable to various games.

3.6. Summary

In this chapter, we proposed a way to represent and implement sub-objectives in our AI as well as a method to ascertain the right set of sub-objectives related to a game. This feature is critical to the development of the proposed GVGPAI.

The presented sub-objective representation is highly dependent on the game data representation which was also described in this chapter. The sub-objective concept used in the current solution must be able to define what is possible to do in a game regardless of the game-play.

In addition the detailed discovery process is actually divided in two sub-processes. The first one is used to generate the list of sub-objective related to the game. The second process's purpose is to ascertain the right set of sub-objectives available in the current agent's position. Using this both processes, the AI is able to adapt to any type of game as well as any situation modification within those games. The discovery process performance was assessed on two different games and the obtained results are very satisfying.

The next step is to determine how to implement the rest of the AI. This implies the choice of the algorithms used for the first and second layers. These layers are used to search for the best strategy and the best sequence of actions respectively. The following chapter presents two different implementations.

Chapter 4

GVGPAI Implementation

4.1. Introduction

This chapter describes two different implementations of GVGPAI. Each one of these implementations is based on the architecture suggested in chapter two and the sub-objective representation and discovery process presented in chapter three. An implementation determines which algorithms are used and how they are interconnected.

Both of the proposed implementations are only composed of the first and second layer described by the architecture. The architecture suggested in this dissertation is complex and requires to be developed step by step in order to avoid error. While the two first layers are essential to the proper functioning of the AI, the third layer can be considered as optional. The third layer's main purpose is to increase the efficiency and adaptability of the AI by handling the emergency cases. However its absence will not prevent the AI to work. Therefore we focused on the two first layers and postponed the development of the third layer.

The first section details the AI's first implementation. It describes the motivations and the preliminary study which led to choose these algorithms and explains their use within the architecture. It ends by stating a list of the drawbacks arising from the chosen implementation. The following section describes the AI's second implementation. It explains the choices made to overcome the flaws observed in the first implementation and how the new algorithms were used within the architecture. Then, the performance of both implementations was assessed and the third section presents the obtained results. Finally the fourth section introduces our main future works.

4.2. First AI Implementation

The implementation of a program defines which algorithms are used and how they are interconnected. This section details the implementation of the first implementation. This one complies with the solutions proposed in the second and third chapters and explains how those concepts are put in practice.

First we explain the motivations that led to this implementation. Then a preliminary study was set up in order to determine which algorithm was the best suitable to achieve these objectives. The third part describes how the chosen algorithms were used in the architecture. Finally a list of the drawbacks arising from this implementation is presented.

4.2.1. Motivations

In video games, the situation, even if in constant evolution generally does not jump from a particular state to a new random one. Therefore, in this first implementation, we decided to implement a first layer keeping improving the existing strategies instead of recreating them at each iteration.

This choice would allow the AI to work in a long-term perspective. Even if at the start of the game the strategies computed are not very good, they will mature as the game progresses and will keep adapting to the game's modifications.

The best suitable type of algorithms for this kind of tasks is the optimization algorithms. The next sub-section presents the result of a study set-up to determine which algorithm was fit to our problem.

4.2.2. Preliminary study

As explained above, the use of an optimization algorithm is recommended for the strategy generation realized by the first layer. There are various optimization algorithms based on different concepts and efficient in different situations. This section details the process we used to determine the algorithm that best fits the first layer's needs.

One of the first layer's objectives is to determine the best sequence of sub-objectives in order to win the game. Those sub-objectives are chosen within a defined list. Therefore the search for the best combination of sub-objectives can be assimilated to a search in a discrete set. We ran a series of experiments to compare various optimization algorithms' efficiency and determine the most fitted to this task.

We conducted these experiments on three major optimization algorithms, the ant colony algorithm [14] and the genetic algorithm [15] [16] and the dynamic particle swarm optimization algorithm [12]. We also decided to test a modified version of each of them. The original version will run the optimization a predefined number of iterations (set to 2000), whereas the modified version can stop the optimization if during a certain interval of time (set to 500 iterations) the solution's improvement rate stays lower than an improvement criterion (set to 0.01).

We used the well know traveler salesman problem (TSP) as environment for our experiments. In this problem, the salesman has to move between a set of city in order to visit each one once. Its objective is to find the best order between the cities to optimize either the time or the distance travelled. We can easily draw a comparison between this problem and our problem. Each city corresponds to a particular sub-objective and the algorithm has to make a discrete search in order to find the best combination.

Each algorithm above-mentioned is tested through and identical series of experiments. These experiments are composed of a set of cities whose positions are randomly generated within a two dimensional space. Each algorithm is conducted trough 100 runs on each one of the experiments and their average computation time and distance travelled are recorded.

The whole process is composed of 30 experiments. The first experiment is composed of 5 cities. The number of cities is increasing by 1 at each experiment until reaching 15 cities. Then, the increasing rate is set to 5 from 15 to 50 cities and to 10 from 50 to 100 cities. The final rate is set to 20. At the 30th experiment we reach a maximum 260 cities. The parameters of the optimization algorithms we chosen based on previous experiment and were set as follows. The genetic algorithms used the mutation rate set to 0.01 and a population of 100 elements. The ant colony optimization algorithms used a population of 100 elements, a pheromone decay rate set to 0.52, an alpha (distance priority) set to 1.3896, beta (pheromone priority) set to 1.5792 and the deposit factor to 10. Finally the dynamic particle swarm optimization algorithms used a population of 100 elements.

The following graphs resumes the results obtained from our series of experiments. The efficiency of the ant colony optimization algorithm (ACO), the dynamic particle swarm optimization algorithm (DPSO), the genetic algorithm (GA) as well as their modified version using an improvement criterion (ACO-IC, DPSO-IC, GA-IC) are all represented in those graphs.

Fig.22 represents the average distance travelled by each algorithm depending on the number of cities. This distance is a good indicator of the quality of the solutions found by the algorithms. The shorter is the distance the better is the quality. Fig.23 is a detailed view of the Fig.22 focusing on the first experiments (from 5 to 12 cities).

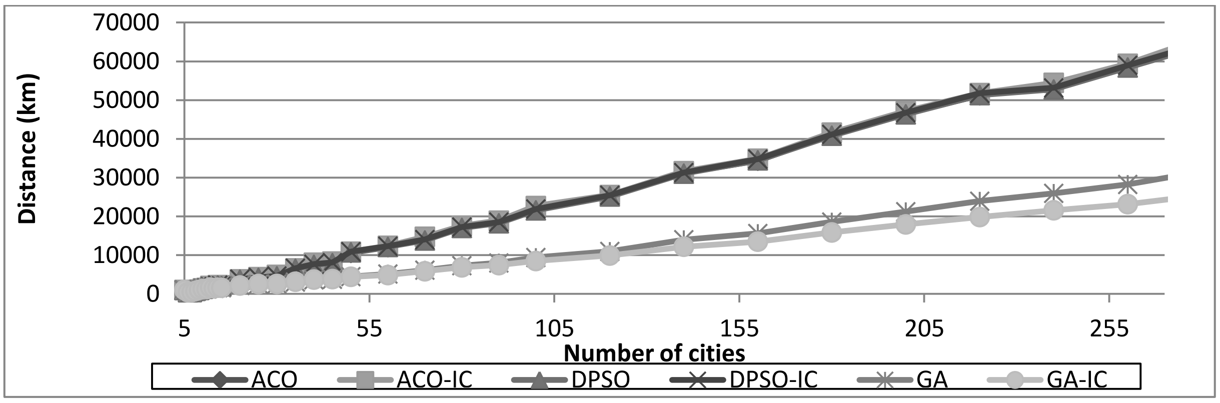


Fig. 22. Average distances travelled.

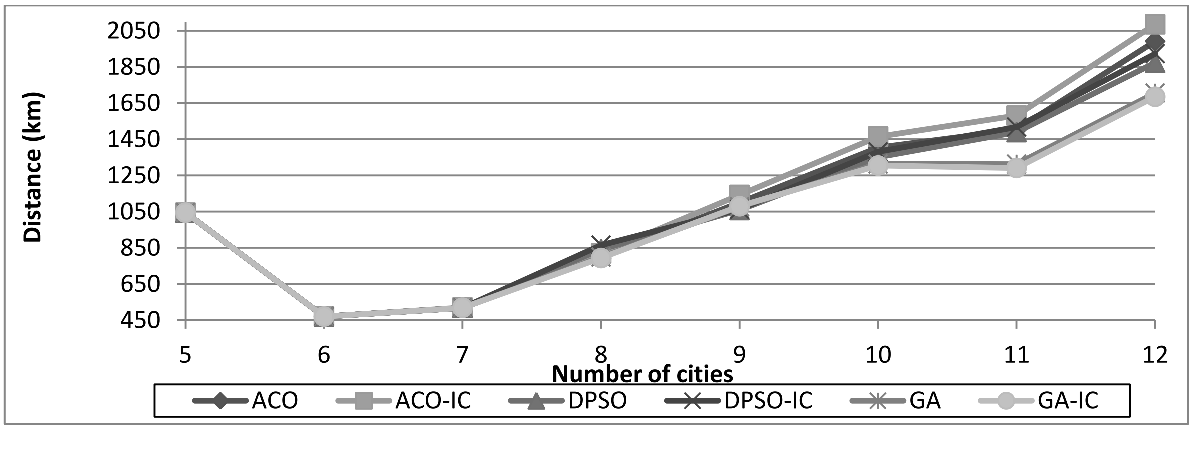


Fig. 23. Average distances travelled (detailled).

Fig.24 represents the average computation time required by each algorithm depending on the number of city. This parameter is as much important as the quality of the solution to determine the algorithm's efficiency. Indeed, finding a perfect solution is irrelevant if it requires a huge computation time. Fig.25 is a detailed view of the Fig.24 focusing on the first experiments (from 5 to 12 cities).

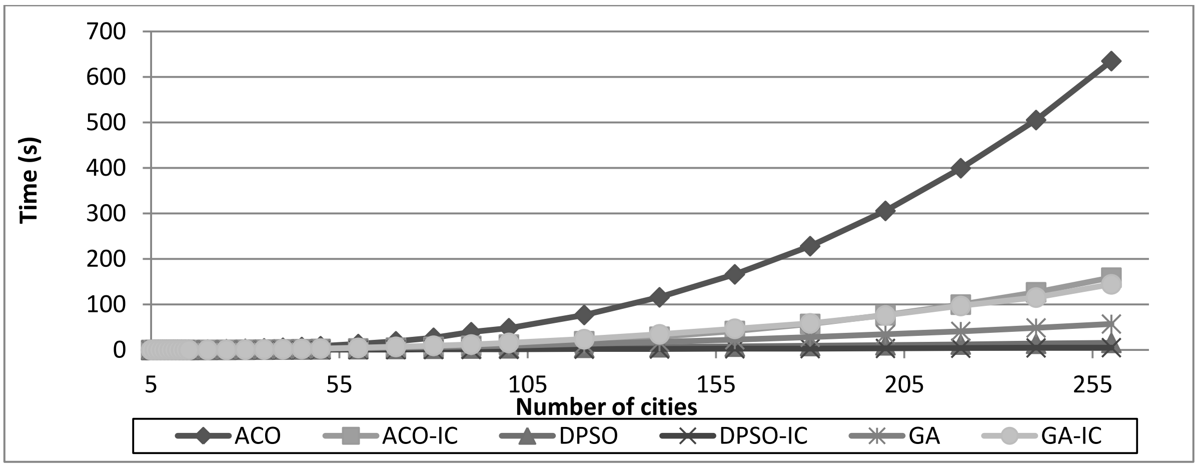


Fig. 24. Average computation time.

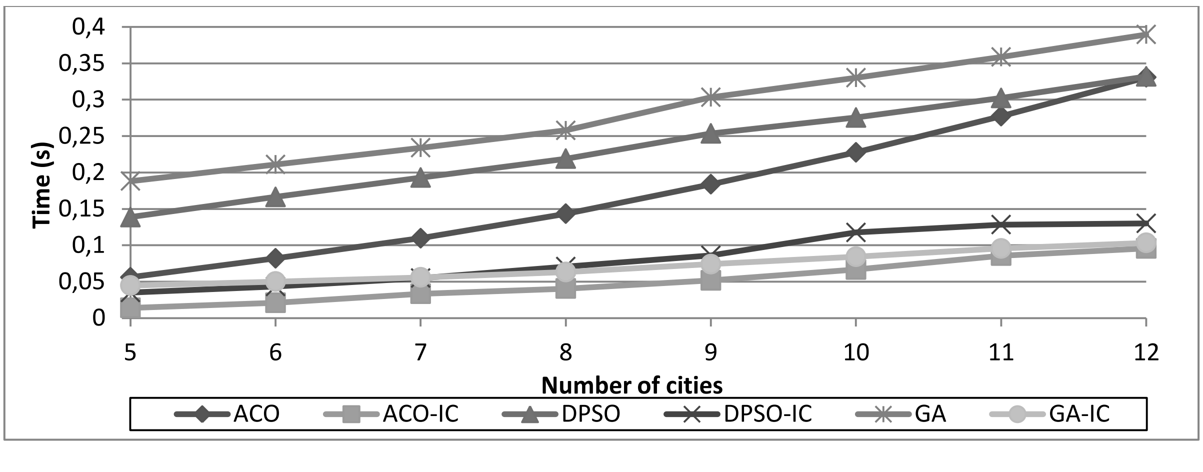


Fig. 25. Average computation time (detailled).

According to the results given by the Fig.22 and Fig.23, the best overall solutions are computed by the genetic algorithm and its modified version. Thought the solutions' quality obtained on little problems are quite the same for both algorithms. One can remark an increasing difference between the qualities found by the two algorithms when the size of the problem exceeds 100 cities. Then, the genetic algorithm with improvement criteria is the best.

The Fig.24 and Fig.25 present mixed results. According to the Fig.24, on problems of more than 100 cities, the dynamic particle swarm optimization algorithm and its modified version are the quickest. However the quality of their solutions isn't satisfying, as shown by the previous results. The next best results are obtained by the genetic algorithm. Then there are the modified genetic algorithm and the modified ant colony optimization. On problems of little size the quickest algorithms are, respectively, the modified ant colony optimization algorithm, the modified genetic algorithm and the standard version of the genetic algorithm.

To refocus on our problem, we are looking for the best-fit to a sub-objectives sequence optimizer. In this context it is unlikely that the number of sub-objectives useful to win the game exceed 100. Therefore, the genetic algorithm with improvement criteria seems to be a good alternative both in terms of quality and efficiency.

Based on the results obtained on this set of experiments we decided to use the genetic algorithm with improvement criteria for the first layer's strategy generation process.

4.2.3. Implementation

The implementation described in this sub-section only includes the two first layers of the suggested architecture. In addition the first layer's first module, which is used to discover the available sub-objectives, is based on the solution detailed in the previous chapter. The first layer's second module purpose is to use the list of available sub-objectives generated by the first module to generate and evaluate multiple strategies. Then it selects the best strategy and sends the next sub-objective to be achieved to the second layer. The first layer's second module described in this implementation is based on a combination of genetic algorithm with improvement criteria and neural network.

During the first iteration, the AI's genetic algorithm creates from scratch a first population of strategies based on random combinations of sub-objectives. These sub-objectives are chosen from the list generated by the first module of the first layer. Then, the GA-IC optimizes that population of strategies until the end of its allowed computation time. Finally the population generated by the GA-IC is recorded in order to be reused in the next iteration.

In order to optimize its population the GA-IC uses a fitness function which determines the quality of a given solution. Because the concept of quality is highly subjective, the fitness function is generally user defined and differs depending on the problem to solve. Therefore, to create a solution suitable for GVGPAI, the fitness function used by the genetic algorithm must be adaptable to every kind of games. We chose to use a neural network. This choice was made in light of the neural network's ability to approximate any functions. Using such a solution would enhance the AI's adaptability. Therefore, each strategy composing the population generated by the GA-IC is sent to this neural network in order to determinate its quality. In addition, the neural network also received the game state in order to put the evaluated strategy in context. Indeed, the quality of a strategy might depend on the agent's current situation. Using a composition of these two blocks of information (strategy and game's state) allows the AI to select a strategy in accordance with the game's state evolution. However, the neural network topology and weights need to be tuned. This operation is performed using NeuroEvolution of Augmenting Topology algorithm (NEAT), which outperforms the best fixed-topology methods.

The NEAT method consists on evolving a population of neural networks using a modified genetic algorithm. This algorithm uses the crossover and mutation features of the GA to modify the weights as well as the topology of the neural networks. Each neural network is evaluated depending on its performance on the game. In our work the quality of a neural network correspond to its average score on a series of 10 game instances. These instances are played using the network to evaluate integrated to the proposed AI. The NEAT method evolves the neural networks using an augmenting topologies policy. This means that the first population of neural networks (randomly generated) is only composed of the input and output layers. Then through the evolution process more neurons and connections are added to the topology. The best neural network evolved is selected and inserted in the final version of our AI. Using the NEAT method, the AI will at first obtain bad results on that game, but as the neural network assimilates the game-play, the results will improve. Finally the next sub-objective is selected. In order to encourage the improvement of the strategies by the GA we decided to achieve all the strategies before to select a new one. If the current strategy is finished the best strategy of the population is selected and its first sub-objective is sent to the second layer. However if it is not finished yet, the AI checks if the current sub-objective has been achieved. If it is the case, then the next sub-objective is sent to the second layer otherwise the current sub-objective is resent. The completion of a sub-objective is evaluated using the game states. For example, if the selected sub-objective is to move the agent to the left, the data corresponding to the agent position in both the previous and the new game state would be compared to ascertain if the agent has moved in the right direction.

The second layer's purpose is to search for the best sequence of actions to achieve the selected sub-objective. Once again the actions choice depends highly on the game. Therefore, to create a solution adaptable to many games we proposed to use a second neural network. This time the neural network's inputs are composed of the selected sub-objective and the game state (for context). The neural networks output defines a list of ids corresponding to the actions to execute. The same method as for the first layer's neural network is used to tune this one.

Using the proposed implementation allows the AI to generate a strategy which even if it is not perfect at first; will be improved during the following iterations. This enables the AI to remain reactive to the potential changes in the game's state and modify the strategy if needed. However, because the strategies and sequence of actions are used as input and output of neural networks respectively, their size must be defined before the networks tuning.

4.2.4. Drawbacks

The performance evaluation is presented afterward. However this sub-section lists some of the drawbacks of the current implementation which have motivated the development of a second implementation:

- Tuning two different neural networks whose qualities are interconnected is a very complex and time consuming task.
- The genetic algorithm requires too much iteration before to obtained strategy good enough to be played. This implies that for some games, the agent either plays using bad strategies or requires lots of computation time before to give back an action to the game.

4.3. Second AI Implementation

The section describes the second implementation. As with the first implementation, this one complies with the solutions proposed in the second and third chapters.

First we explain the motivations that led to this implementation. Then, the second sub-section describes how the algorithms were used in the architecture.

4.3.1. Motivations

The first version encountered a set of drawbacks which have encouraged the development of a second implementation.

First, because both layers were composed of neural networks whose qualities are interconnected, their tuning process requires a considerable computation time. In order to overcome this problem we decided to change the solution used by the second layer. With only the first layer's neural network left the tuning process would be quicker.

The second modification made to the AI implementation is related to the strategy generation process. While the evaluation of these strategies is still performed using a neural network, their generation is operated using a different algorithm. This choice was made in light of the impossibility to use the genetic algorithm on games requiring quick reactions. In addition we realized that some games might be entirely random thereby forbidding the use of progressively improved strategies.

Both elements were replaced using MCTS components. However, as explained in the introduction the MCTS is weak against game with deep decision tree. Therefore, these MCTS components were modified so they do not encounter such drawback. These modifications are explained in the following sub-section.

4.3.2. Implementation

This section describes the implementation of the first layer's second module and the second layer. Both layers include MCTS components. We detail the adjustments made to these components to fit them in our AI.

As explained before the second module of the first layer uses the previously computed list of available sub-objectives to generate and evaluate a list of strategies. This process is based on a MCTS component coupled to a neural network. The MCTS is used to generate various sequences of sub-objectives and the neural network evaluates them. The first layer uses a plain UCT which is a MCTS using UCB1 as proposed by L. Kocsis and C. Szepesváris [17]. The particularity of this MCTS component is that instead of working with actions, it works on sub-objectives. The sub-objectives are picked from the list of available sub-objectives and the MCTS built a tree upon different sequences of sub-objectives. Our AI uses strategies of predetermined size. Therefore, the tree deepness never exceeds that size. The size of the decision tree is reduced drastically and is linked mainly to the game complexity (number of sub-objectives): not to its length. Consequently this MCTS component does not suffer of the standard MCTS's weakness. In addition, as explained above, a single sub-objective can require more than one action to be achieved. There might be more than one way to complete it. Therefore, the first layer's MCTS component cannot use standard simulation to obtain a reward. Instead, we generate rewards using a neural network. First a strategy, a sequence of sub-objectives, is selected by the tree policy. Then to evaluate that strategy, the neural network takes as inputs the current game's state (formatted as a list of float) and the strategy to evaluate. If the strategy generated by the MCTS is too small to entirely fill the input layer, random sub-objectives are added. Afterward, the network's output is a single number, between 0 and 1, corresponding to that strategy's reward. Finally that reward is sent back to the MCTS and back propagated through the tree. Fig.26 represents the process explained above.

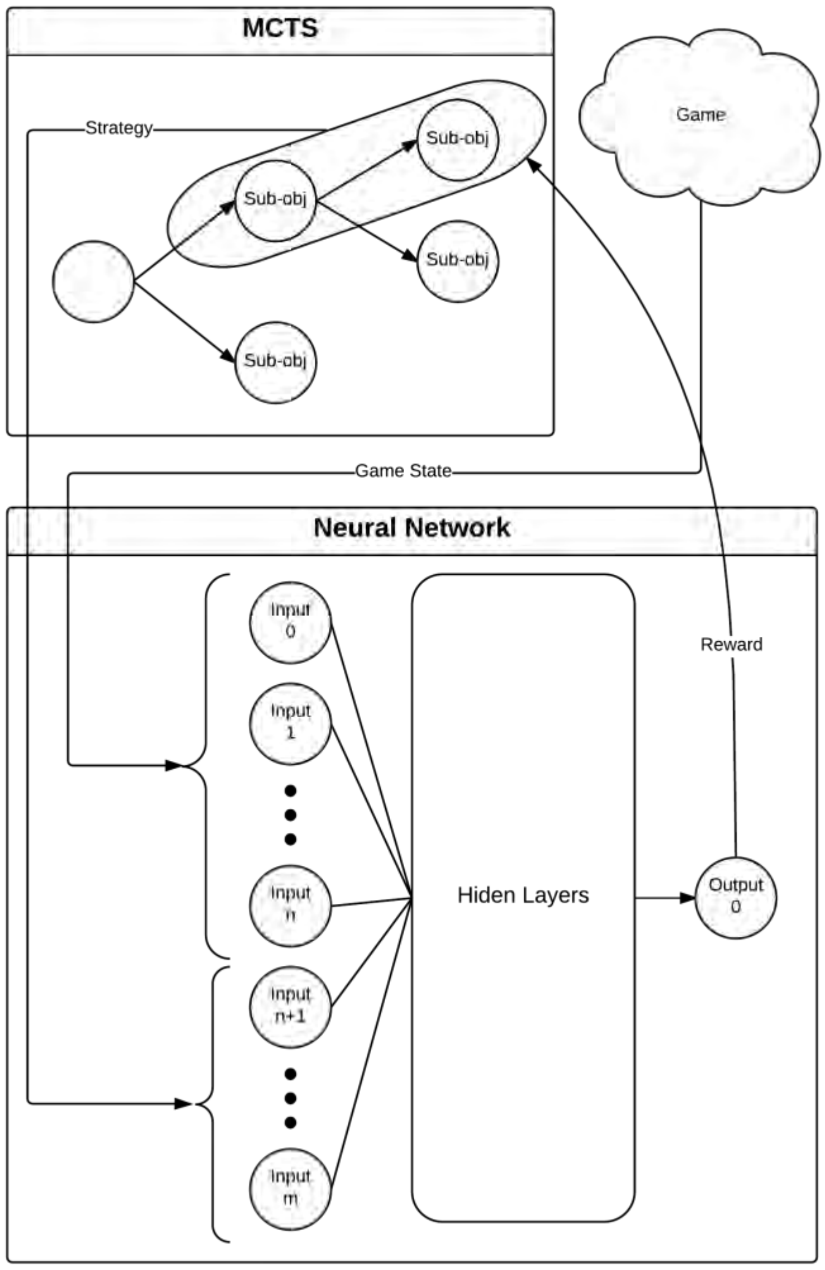
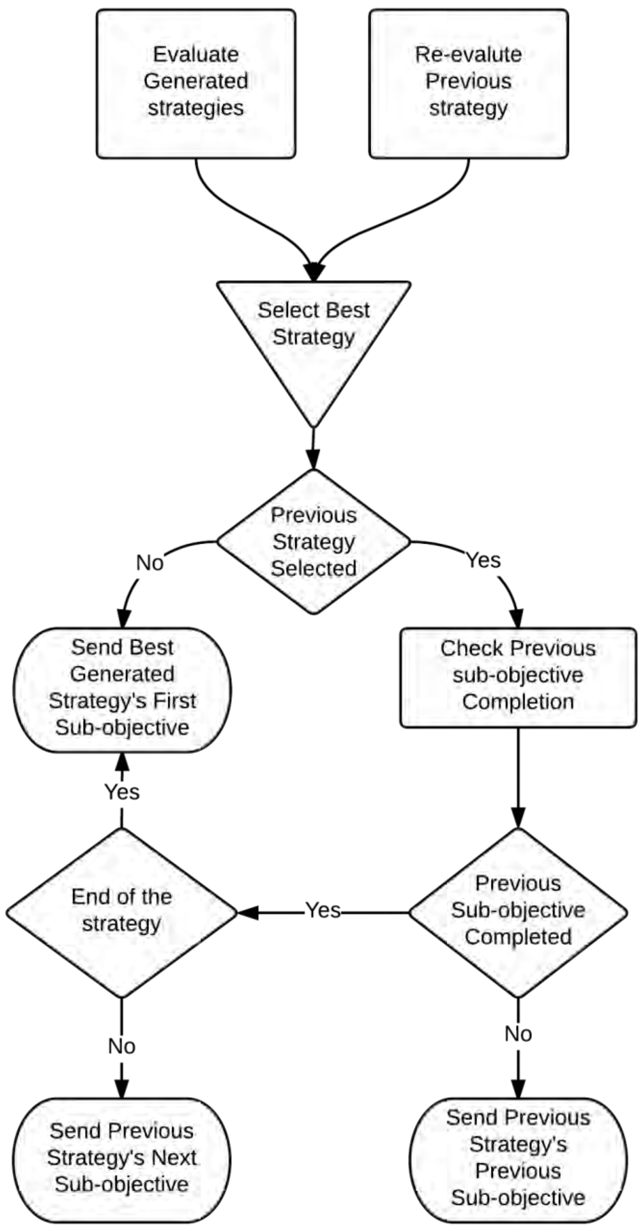


Fig. 26. Representation of the reward generation process.

As with the first implementation, the neural network is generated using NeuroEvolution of Augmenting Topology algorithm (NEAT) and it will assimilate the game-play. The MCTS runs a determined number of iterations. The most rewarded sequence of sub-objectives is selected.

Finally the quality strategy which is currently being used is recalculated using the neural network. It is compared to the newly computed strategy. If the new strategy has a better quality, then it replaces the current one. Its first sub-objective is then sent to the second layer.

However, if the current strategy is still better, then the AI checks if the current sub-objective has been achieved. If it is the case, then the next sub-objective is sent to the second layer otherwise the current sub-objective is resent. Fig.27 depicts the sub-objective selection process.



Evaluate Generated strategies
Re-evalute Previous strategy
Select Best Strategy
Previous Strategy Selected
No
Yes
Send Best Generated Strategy's First Sub-objective
Check Previous sub-objective Completion
Yes
End of the strategy
Yes
Previous Sub-objective Completed
No
No
Send Previous Strategy's Next Sub-objective
Send Previous Strategy's Previous Sub-objective

Fig. 27. Next sub-objective selection process.

The second layer's objective is to find the best sequence of actions to accomplish the sub-objective sent by the first layer. The actions are selected from the list proposed by the game. In the second implementation, the second layer is also based on a MCTS component. In this layer too, we use a plain UCT which is a MCTS using UCB1 as proposed by Kocsis and Szepesv áris. Each time the second layer is called, a new tree is built. The tree expands depending on the available actions. During the simulation step, the AI uses the simulation tools proposed by the game to simulate the future game state. These simulations are performed on a copy of the game.

The particularity of the second layer is that instead of searching for the best sequence of actions to win the game, it seeks the best sequence of actions to achieve the selected sub-objective. The sub-objective completion is checked a first time at the end of the MCTS's expansion step. If the sub-objective is not accomplished yet, then the MCTS starts the simulation step. During the simulation step, actions are selected randomly. The sub-objective completion is checked every iteration. If the sub-objective is achieved during the tree policy the reward is set to 1. If it is achieved during the simulation step, the reward is set to 0.5. However, if the game ends before the accomplishment of the sub-objective, the reward is set to 0.

The search specifically examines accomplishment of a particular sub-objective instead of winning the game. Therefore, it is generally solved with short planning and is drastically reducing the decision tree size. This MCTS component does not suffer from the weakness faced by the standard MCTS.

This MCTS component runs a determined number of iterations. When the process is finished, the action corresponding to most rewarded root child is selected and is sent back to the game.

4.4. Performance Evaluation

The GVGPAI presented in this dissertation was developed to use a more human-like mode of thinking and is based on the two following key points:

- The ability to determine the set of sub-objectives related to a particular problem.
- The ability to separate the strategy and action selection processes.

The third chapter which is dedicated to the sub-objective representation and discovery process showed very satisfying results. In addition, the architecture as the two AI implementations described in the first and fourth chapters clearly separate the strategy and action selection processes.

However, in order to prove that a more human-like mode of thinking resulted from this and that the proposed solutions are efficient enough to compete with other GVGPAI we set up three experiments. Each one is based on the two the games described in chapter three and is performed on both of the implementations.

This section analyzes the experimental results. First we assess the ability of the AI to use a more human-like way of playing. Then we evaluate their efficiency by comparing their score with other GVGPAI. Finally we compare the computation time required by these solutions.

4.4.1. First Experiment

This sub-section evaluates the ability of the proposed AI to use a more human-like way of playing. This point is an important feature of this paper.

In the third chapter, we assessed the ability of our AI to ascertain the right set of sub-objectives related to the game. For this purpose we extracted the list of sub-objectives generated by the AI on the two games presented previously. The results showed that the proposed AI was able to ascertain all of sub-objectives related to the game.

However another criterion to assess the ability of the AI to use a human-like mode of thinking is the stability of its choices. If the AI is able to maintain its most of the selected strategies until their completion, it would result in a seemingly human way to play. In order to evaluate that, we measured how many strategies were entirely achieved.

From this point, we forced the AI to only use the sub-objectives related to the object "player". This modification is selected using a parameter which is set before the program is launched. This choice was made based on three factors. First, using the third chapter's experimental results, we realized that the main features of a game are directly related to the player. Then the neural networks used by the proposed AI are evolved using the NEAT method. This one requires running many game instances and is very time consuming. Finally the sub-objective non-related to the player would have been rejected by the neural network during its evolution because they are not essential to win the game. This rejection would have resulted by the neural network attributing a bad quality to all the strategies including sub-objectives non related to the player. Therefore this choice did not change the final results of the experiments but only allowed to accelerate the neural network generation process. Furthermore, the first implementation of our AI generates strategies of composed of 10 sub-objectives, more fit to be improved by the genetic algorithms. While the second implementation of our AI generates strategies composed of a single sub-objective. In future researches we aim to use longer strategies and to create a method able to better determine the irrelevant sub-objectives in order to improve the quality of the generated strategies. Table 5 gives the percentage of strategy achieved.

TABLE V. PERCENTAGE OF STRATEGY ACHIEVED

	Card Game	**Motif Game**
1st implementation's strategies completion rate	100%	100%
2nd implementation's strategies completion rate	90%	80%

The results given by the Table 5 shows that our AI was able to achieve most of the selected strategies. The first implementation achieved 100% of the selected strategies because of its particular implementation (all strategies are always achieved in order to encourage the improvement of the population). However, the second implementation allows selecting a new strategy if that one is considered better (see Fig.27). Nevertheless the AI based on the second implementation completed over 80% of its strategies. This means that, most of the time, the AI kept the same goal until its completion before to choose a new one. This suggests that our solution acted in a correlated way, following a self-determined strategy until its end, such as a human would do.

4.4.2. Second Experiment

To prove that our proposal is able to compete with other GVGPAI, we compare on both games the scores obtained by the following AI:

- Monte Carlo Tree Search (UCB1) based AI
- NEAT based AI
- Random AI
- 1st Implementation
- 2nd Implementation

The NEAT based AI and our solutions were trained on a population of 200 networks before to be tested.

Table 6 shows the average scores (on 100 games) of these five AI. The scores are calculated as explained by the games' description in chapter 3. The higher the score, the better the AI has performed.

TABLE VI. AVERAGE GAME SCORE

	Card Game	**Motif Game**
MCTS	4.44	9
NEAT	8.48	6.14
Random	2.66	4.36
1st Implementation	7.54	4.12
2nd Implementation	9.16	4.62

In addition, Fig.28 and Fig.29 depict the boxplots obtained using both games' score.

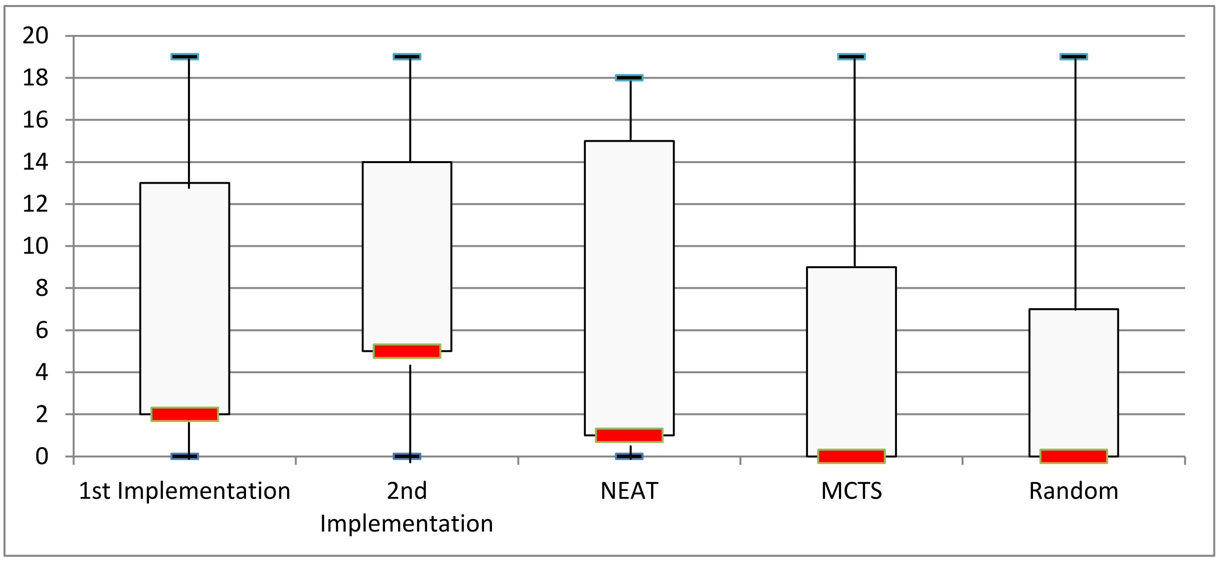


Fig. 28. Boxplots obtained on the card game.

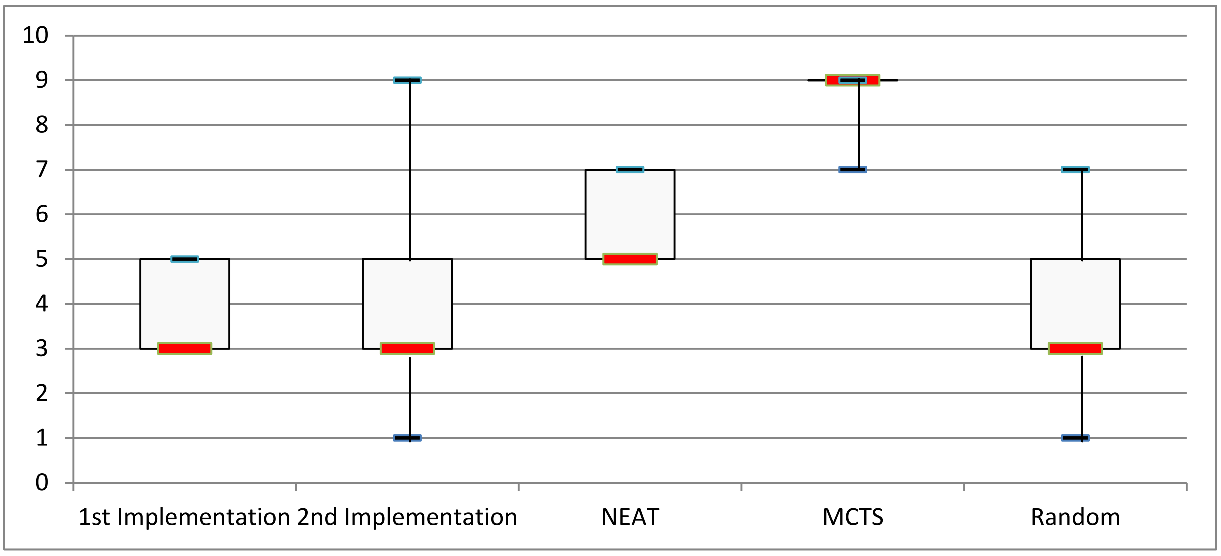


Fig. 29. Boxplots obtained on the motif game.

The results obtained on the card game are satisfying. Our first implementation obtains a score a little lower than the NEAT while our second implementation a score a little higher. However, the card game is highly random the MCTS obtained a lower average score. This suggests that an improvement was made between the two implementations. In addition it proves that's the proposed solution was able to learn how to play to that game and is efficient enough to compete with other GVGPAI in some games.

These results are confirmed by the boxplots obtained using the card game's scores. We remark that the median score obtained by our second implementation is higher than all the others (including the NEAT's median).

The results obtained on the motif game are less conclusive. This game is deterministic and has a shallow decision tree (only 5 iterations), which explain why the MCTS obtained an excellent score. However our solution obtained an average score comparable to the random AI. Nevertheless the corresponding boxplots shows that our second implementation has greater maximum score than the random AI. This suggests that our AI slightly learned how to play this game but requires more training. The boxplot obtained by the first implementation shows worst results than the random AI and second implementation. This could due to a too slow improvement of the strategies, thus resulting in failure of the AI on the game. This suggests that a solution progressively improving its strategies is not applicable to quick games such as this one.

These experimental results suggest the ability of our proposal to adapt to various games and to compete with other GVGPAI. However it also showed that because of the complexity of its implementation the learning process of the proposed solution require more computation time than the other solutions. This particularity is directly related to the ability of our AI to self-generate its own domain-specific knowledge. In addition these results validated the choices made for the second implementation by presenting a clear improvement between these two implementations.

4.4.3. Third Experiment

In the third experiment we assess the performance of our solution by comparing the computation time required for a single game iteration. Table 7 shows the average computation time(millisec) required by all five GVGPAI solutions on the card games. The results presented are the average value obtained over 100 runs.

TABLE VII. AVERAGE COMPUTATION TIME

	Card Game
MCTS	3.408
NEAT	0.005
Random	<1 millisec
1st Implementation	118.45
2nd Implementation	1.029

First, these results prove once again the improvement made between the first and second implementation. Then we observe that the computation time required by our solution is positioned between the MCTS and NEAT based AI which suggest that it come to a range of computation time which can compete with other solutions. Finally, traditionally in video games an iteration is performed every 10 milliseconds, based on that standard our solution is usable on such games.

4.5. Future works

Future works for this study focus on three points:

- First, some effort should be put into reducing the computation time required by the training process of the proposed AI. The neural networks used to assess the generated strategies are tuned using the NEAT method. These methods as well as the other deep learning method are known to require a long computation time in order to obtain viable results. However reducing the number of usable sub-objectives should simplify the process and reduce the time required by the learning process.
- Therefore, more works should be done on the sub-objectives. The current method used to evaluate their relevancy is quite simple and a more precise assessment of the sub-objectives usefulness would lead to a global improvement of the strategies generated and as suggested above an acceleration of the learning process.
- Finally the third layer must be implemented. Until then only the first and second layers were implemented. These layers are essential to the proposed AI functioning and we chose to develop them apart from the third layer in order to obtain a better assessment of their efficiency. However, the third layer may improve the global performance of the proposed solution and should be developed from now on.

4.6. Summary

This chapter presented two GVGPAI implementations based on the architecture and concepts described in the previous chapters. The second implementation was build depending on the flaws discovered in the first one and their performance evaluation proved the validity of the taken choices. In addition the results obtained during the performance evaluation proved their adaptability to many games. Nevertheless, it also shown that the complexity of the proposed solution resulted in a longer training phase. Finally the performance evaluation proved the ability of our AI to use a more humane way of playing which was one of the main target set by this study.

Chapter 5
Conclusion

To conclude this dissertation, this study's target was to develop a multilayer GVGPAI using a human-like way of playing. This problem was divided in two key features. First the AI has to be able to learn itself the games' characteristics, then the strategy and action selection processes has to be separated.

First, we described an architecture based on three layers. The first one ascertains the sub-objective related to the game and uses them to generate multiple strategies. The second layer searches for the best sequence of actions in order to achieve the selected sub-objectives. Finally the third layer is responsible to handles emergency situations. Any AI developed based on this architecture would comply with the second key feature enounced. The results obtained on the geometry friend's competition with an AI partially based on the proposed architecture were encouraging.

Then we proposed a solution to ascertain the set of sub-objectives related to the game. Sub-objectives a core concept of this study and in order to comply with the first key feature our AI has to be able to ascertain them. Sub-objectives represent what can be done in a game and the sub-objectives used in this study were specifically designed to be able to represent any sub-objectives regardless of the game-play. The discovery process presented in this dissertation is actually divided in two sub-processes. The first one is used to generate the list of sub-objective related to the game. The second process's purpose is to ascertain the right set of sub-objectives available in the current agent's position. Because the set of sub-objectives available in a specific situation may not contain all the sub-objectives usable in the game, the identification process increases the efficiency of the proposed solution.

In addition, two different implementations for GVGPAI were proposed. Both of them are based on the architecture and concepts described in this dissertation but the second implementation was built depending on the flaws discovered in the first one. A performance evaluation was done on two experiments, each one based on a different game. Overall the obtained results were satisfying. It proved   the adaptability of the proposed solution to many games and its ability of our AI to use a more humane way of playing which was one of the main target set by this study. Nevertheless, it also shown that the complexity of the proposed solution resulted in a longer training phase and a loss of performance on some games.

In conclusion, the main targets set by this study were reached but more works could be done in order to improve the current solution. First of all the learning phase should be shortened. Then a better assessment of the sub-objective must be implemented. Finally the third layer must be implemented. These three points constitutes the main target of our future work.

REFERENCES

[1] A. Nareyek, “AI in computer games,” Queue, Vol.1, No.10, pp. 58-65, February 2004.

[2] D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche,... & S. Dieleman, “Mastering the game of Go with deep neural networks and tree search,” . Nature, 529(7587), 484-489, 2016.

[3] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, “Monte-Carlo tree search: a new framework for game ai,” AIIDE, 2008.

[4] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, et al., “A survey of Monte Carlo tree search methods,” Computational Intelligence and AI in Games, IEEE Trans. on, Vol.4, No.1, pp. 1- 43, March 2012.

[5] J. Méhat and T. Cazenave, “Combining UCT and nested Monte Carlo search for single-player general game playing,” Computational Intelligence and AI in Games, IEEE Trans. on, Vol.2, No.4, pp. 271-272, 2010.

[6] K. O. Stanley and R. Miikkulainen, “Evolving neural networks through augmenting topologies,” Evolutionary Computation, Vol.10, pp.2, pp. 99-127, 2002.

[7] S. Samothrakis, D. Perez-Liebana, S. M. Lucas, and M. Fasli, “Neuroevolution for general video game playing,” In Computational Intelligence and Games (CIG), pp. 200-207, August 2015.

[8] M. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone, “A neuroevolution approach to general atari game playing,” Computational Intelligence and AI in Games, IEEE Trans. on, Vol.6, No.4, pp. 355-366, 2014.

[9] E. Powley, D. Whitehouse, and P. Cowling, “Monte Carlo tree search with macro-actions and heuristic route planning for the Physical Travelling Salesman Problem,” Computational Intelligence and Games (CIG), 2012 IEEE Conf. on, pp. 234-241, September 2012.

[10] D. Perez, P. Rohlfshagen, and S. Lucas, “The physical travelling salesman problem: WCCI 2012 competition,” Evolutionary Computation (CEC), 2012 IEEE Congress on, pp. 1-8, June 2012.

[11] M. Buro, “Real-time strategy games: A new AI research challenge,” IJCAI, pp. 1534-1535, 2003.

[12] B. Vallade and T. Nakashima, “Improving particle swarm optimization algorithm and its application to physical traveling salesman problems with a dynamic search space,” In Applied Computing and Information Technology, Springer International Publishing, pp 105-119, 2014.

[13] B. Vallade and T. Nakashima, “A new approach of path-finding by possibilities search,” Proc. of Joint 7th International Conference on Soft Computing Intelligent Systems and 15th International Symposium on Advanced Intelligent Systems (SCIS&ISIS2014), pp380-385, 2014.

[14] M. Dorigo and M. Birattari, "Ant colony optimization," In Encyclopedia of Machine Learning Springer US, pp. 36-39, 2010.

[15] C. Reeves, "Genetic algorithms," In Handbook of Metaheuristics, Springer US, pp. 55-82, 2003.

[16] J. John Grefenstette, "Genetic Algorithms and Their Applications," Proc. of the Second International Conference on Genetic Algorithms, Psychology Press., 2013.

[17] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," In Proc. Eur. Conf. Mach. Learn., Berlin, Germany, pp. 282–293, 2006.

PUBLISHED PAPERS BY THE AUTHOR

[1] B. Vallade, A. David, and T. Nakashima, "Proposition of a three layers architecture for adaptable artificial intelligence," Proc. of Joint 7th International Conference on Soft Computing Intelligent Systems and 15th International Symposium on Advanced Intelligent Systems (SCIS&ISIS2014), pp. 247-252,December 2014.

[2] B. Vallade, A. David, and T. Nakashima, " Three layers framework concept for adjustable artificial intelligence," Journal of Advanced Computational Intelligence and Intelligent Informatics (JACIII), Vol.19, No.6, pp. 867-879, October 2015.

[3] B. Vallade, A. David, and T. Nakashima. "Game's strategies identification for adaptable gaming artificial intelligence," Proc. of The 16th International Symposium on Advanced Intelligent Systems (ISIS2015), pp. 696-712, November 2015.

[4] B. Vallade, A. David, and T. Nakashima. "Game strategy decision module based on GA and neural network," In Journal of Computer Games and Communication, Vol.1, No.1, pp.1-12, 2015.

[5] B. Vallade, and T. Nakashima. "Sub-objective based general video game playing artificial intelligence using Monte Carlo tree search and neural network," Computation Intelligence and Games Conference (CIG2016), 2016 (submitted).